WhisperTest: A Voice-Control-based Library for iOS UI Automation

Zahra Moti
Radboud University
Nijmegen, The Netherlands
zahra.moti@ru.nl

Tom Janssen-Groesbeek*
Radboud University
Nijmegen, The Netherlands
tom.janssen-groesbeek@ru.nl

Steven Monteiro*
University of Twente
Enschede, The Netherlands
s.c.monteiro@student.utwente.nl

Andrea Continella
University of Twente
Enschede, The Netherlands
a.continella@utwente.nl

Gunes Acar
Radboud University
Nijmegen, The Netherlands
g.acar@cs.ru.nl

Abstract

Dynamic analysis and UI automation are essential for scalable detection of privacy leaks, vulnerabilities, and malicious code in mobile apps. While the Android ecosystem offers a variety of tools, options for iOS apps are limited and require either access to the app source code or jailbreaking the test device. To address this gap, we introduce WhisperTest, an open-source iOS UI automation library that operates without jailbreaking. WhisperTest is based on a newly designed approach that leverages Apple's Voice Control accessibility feature to interact with app or system UIs via text-to-speech. During interactions, WhisperTest monitors the device system logs in real time and scrapes the UI via screenshots and accessibility audits to recover app state changes. We demonstrate WhisperTest's capabilities through a diverse set of tasks, including a web privacy measurement and a fully-automated dynamic analysis of 200 child-directed iOS apps. To overcome the challenges of automating apps with diverse UI designs, WhisperTest optionally integrates multimodal large language models to reason about context and interact with system permission prompts, consent dialogs, subscription prompts, and age gates. Our exploratory analysis of children's apps uncovers widespread use of third-party tracking, limited recognition of user consent, and unencrypted HTTP requests. Overall, we show that WhisperTest enables scalable dynamic analysis of iOS applications across diverse tasks, contributing to a safer and more transparent mobile ecosystem.

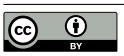
CCS Concepts

• **Security and privacy** → *Network security*; *Software and application security*; Systems security;

Keywords

iOS, mobile testing, automation, privacy, security

*Both authors contributed equally to this research.



CCS '25, Taipei, Taiwan

ACM ISBN 979-8-4007-1525-9/2025/10
https://doi.org/10.1145/3719027.3765183

ACM Reference Format:
Zahra Moti, Tom Janssen-Groesbeek, Steven Monteiro, Andrea Continella, and Gunes Acar. 2025. WhisperTest: A Voice-Control-based Library for iOS UI Automation. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 24 pages. https://doi.org/10.1145/3719027.3765183

1 Introduction

Automated testing of apps and websites is a crucial means for empirical research in security, privacy, accessibility, and performance, among others. Beyond academic and industrial research, quality assurance processes that guarantee software reliability and detect bugs largely rely on automation and testing libraries. Compared to browser automation libraries such as Selenium, Playwright, and Puppeteer, there are a limited number of options for user interface (UI) automation for mobile apps, even less so for Apple's notoriously closed software ecosystem. Many of the existing iOS UI automation libraries, such as Google's EarlGrey [1], allow developers to only test their own apps by relying on Apple's own UI testing framework, XCUIAutomation [2]. While Appium XCUITest Driver [3] can be used for testing any iOS app, it requires a macOS host computer, a complicated setup, and it suffers from instability [4]. This leaves a gap for a cross-platform, open-source, and low-dependency automation library that can interact with arbitrary apps, without modifying the app itself or jailbreaking the test device. With jailbreak tools becoming scarce and some containing malicious code, their use also raises concerns about forensic integrity [5].

To address this gap, we present WhisperTest, an open-source and versatile iOS app automation library. WhisperTest relies on a newly designed approach that combines text-to-speech and iOS's Voice Control accessibility features to send navigational commands such as tap, scroll, and swipe as voice commands. At run-time, our library recovers state updates from the device by monitoring system logs, screenshots, and accessibility audits.

In its core, WhisperTest provides a device object that can be used to install, launch, enumerate, and interact with apps or system menus, similar to Selenium's WebDriver [6]. The device object can also be used to capture network traffic (without decryption) and gather information about the device's state—such as what is displayed on the screen—using optical character recognition (OCR),

icon detection, and accessibility audits. WhisperTest exposes real-time system logs to developers, providing a rich source of low-level debugging information about the device, OS, and processes.

WhisperTest is designed to be extensible: it can be integrated with a decision system to guide app interactions, including Large Language Models (LLMs), UI agents or simple rule- or trigger-based logic. To showcase WhisperTest's adaptability in challenging scenarios, we design an LLM-based navigation pipeline tailored to perform privacy-related measurements in 200 apps targeted at children[1], enabling the detection of advertisements (ads), third-party trackers, and unencrypted data. We choose these apps because they are particularly difficult to automate given their animated, cartoonish interfaces and limited accessibility, and the frequent appearance of consent dialogs and age gates. We also tested WhisperTest's applicability to broader app categories by conducting additional experiments on 50 popular non-children's apps. A larger-scale, comprehensive demonstration of the applicability of WhisperTest across different app categories is left out of scope. In addition, we demonstrate how WhisperTest can be used for web automation and for automating iOS system menus and features. Our measurement reveals that children's apps frequently incorporate third-party tracking, provide limited recognition of user consent, and occasionally rely on unencrypted HTTP requests—behaviors that only become observable after active UI interaction, further motivating a tool like ours.

In summary, we make the following contributions:

(1) We present WhisperTest, an iOS automation library that leverages Apple's Voice Control to automate iOS devices. WhisperTest does not require jailbreaking and can interact with arbitrary apps, websites, and iOS system menus.
(2) To show the versatility and extensibility of WhisperTest, we design and integrate an LLM-based navigation approach and conduct a survey of tracking and ads on 200 iOS apps targeted at children.
(3) We demonstrate WhisperTest's capabilities by presenting additional use cases, including an illustrative web measurement study that compares tracking on news and misinformation websites.

WhisperTest source code and additional materials are publicly available at: https://github.com/iOSWhisperTest.

2 Background and Related Work

2.1 Mobile Automation Tools

Mobile automation has become essential for testing and analyzing app behavior for purposes including security [7–9], privacy [4, 10, 11], and accessibility [12]. Traditional testing and automation tools such as Android Debug Bridge (`adb`) [13] and Appium [14] have been instrumental in automating mobile interactions and have been used in many prior studies [4, 15, 16]. Beyond UI interaction, dynamic binary instrumentation frameworks such as Frida allow researchers to reverse engineer mobile apps by adding instrumentation code at runtime [17], while tools such as `apktool` allow repackaging mobile apps to enable TLS decryption and bypassing certificate pinning [18, 19].

Only a small subset of these libraries can be used for UI testing, which requires the ability to scrape the screen contents and send events such as taps, scrolls, or swipes. On iOS, most existing solutions are restricted in functionality due to Apple's security model [20]. Below, we give an overview of prominent mobile automation tools with a focus on the iOS platform.

Keep It Functional (KIF) [21] is an open-source iOS UI testing framework built on Apple's XCTest. KIF automates interactions via accessibility labels, requiring tested apps to be fully accessible. Since the test code must be embedded while building the app, KIF only supports white-box testing of apps under developer control. KIF is macOS-only and supports iOS versions up to 13.

EarlGrey [1], developed by Google, is another XCTest-based framework offering advanced synchronization and interaction capabilities, but is also designed for internal app testing. According to its documentation, EarlGrey supports iOS versions up to 15.

iOS Development Bridge (`idb`) [22] is a command-line interface designed to automate interactions with iOS simulators and devices. `idb` aims to provide a consistent interface for tasks such as installing apps, managing simulators, and retrieving device logs. `idb` depends on Xcode and does not inherently support UI automation for third-party applications.

Calabash [23] was an open-source mobile automation tool for testing native and hybrid apps. It supported iOS and Android but has been officially deprecated and unmaintained since 2017.

Appium [14] provides a flexible solution for automating both Android and iOS apps using the WebDriver protocol and has been widely adopted in research for automating UI interactions at scale [4, 24–28]. On iOS, Appium relies on the Appium-XCUITest driver [3], which builds on Apple's XCUITest framework and WebDriverAgent [29]. Appium-XCUITest driver exclusively runs on macOS, relies on Xcode integration, and found to be unreliable [4, 30–32].

NoSmoke [33] is a UI automation framework designed to support automated testing and dynamic analysis of Android and iOS apps [33]. Similar to Apium, NoSmoke is based on the WebDriver protocol and relies on Apple's XCTest framework for iOS automation, requiring macOS to operate. However, NoSmoke has not received updates in over three years.

`pymobiledevice3` [34] is a Python-based library that communicates with iOS devices over USB or network using Apple's internal protocols. It enables access to a range of low-level services such as `lockdownd` (device pairing), `afc` (filesystem access), `syslog` (live logging), and `pcapd` (packet capture). These capabilities are exposed through an open Python framework and do not require jailbreaking, making the platform practical for real-world app analysis. Tools similar to `pymobiledevice3` include `go-ios`[35], a Golang-based CLI for iOS device interaction, and `libimobiledevice` [36], a cross-platform library offering similar access to iOS internals through open-source implementations of Apple's device protocols.

Comparison with Existing Tools Unlike tools used in most prior works, the WhisperTest can run on almost any operating system, not just macOS. WhisperTest operate on non-jailbroken iOS devices, enabling the testing of many apps that with anti-debugging measures and refuse to run on jailbroken devices—a critical feature given that the latest two major iOS versions (17 and 18) currently

[1]We consider apps targeted at children as those with a content rating of 4+ on the App Store, based on Apple's age rating system. See: https://developer.apple.com/help/app-store-connect/reference/age-ratings/

lack any public jailbreaks. Unlike tools that depend on outdated platforms or unofficial App Store interfaces, WhisperTest can install apps using voice commands directly from the App Store (§ 4.3). This alternative approach proved resilient when IPATool, a commonly used tool to download and install iOS apps, became temporarily unusable for months due to undocumented backend changes by Apple [37, 38]. By leveraging Raspberry Pi and a USB On-The-Go setup [39], WhisperTest experimentally supports silently sending voice commands and emulating peripherals such as microphones, keyboards, and mice. Its lightweight, modular design accommodates various interaction strategies, including the LLM-guided navigation that we demonstrated in §5.1. WhisperTest also has certain shortcomings compared to existing tools. For instance, it cannot decrypt the network traffic and is slower than existing tools due to its use of voice commands. Despite these limitations, we believe WhisperTest is a versatile, extensible and cross-platform library that supports a diverse set of analysis and testing scenarios. Note that WhisperTest is designed for iOS automation and by "cross-platform" we refer to support for different operating systems (macOS, Linux, Windows) on the controller machine (Figure 1).

WhisperTest uses `pymobiledevice3` for low-level communications with test devices. While `pymobiledevice3` provides a solid foundation, WhisperTest significantly expands its functionality. For example, while `pymobiledevice3` exposes a `perform_press` method to interact with accessible UI elements, the method only works with apps built in debug mode, limiting its use for testing arbitrary apps [40]. As mentioned in § 3.1, WhisperTest circumvents these constraints by relying on accessibility and voice-based mechanisms that work reliably across real-world devices. Table 1 summarizes how WhisperTest compares to other iOS automation tools across key dimensions, showing that it offers competitive capabilities with minimal constraints and dependencies.

2.2 Voice Control

Apple's Voice Control [41] is a built-in accessibility feature that allows hands-free interaction with Apple devices through predefined voice commands for navigation, control and dictation, among others. Beyond this, it supports custom commands and integration with the Shortcuts app, enabling users to automate multistep actions and simulate gestures via voice.

WhisperTest leverages these capabilities to enable flexible and programmable automation. For apps lacking accessibility support—particularly cartoonish or visually complex interfaces—WhisperTest uses Voice Control's overlay commands "Show Grid" which divides the screen into numbered sections for precise tapping (see Figure 2). These overlays allow the system to interact with arbitrary UI elements, even when accessibility data is unavailable, broadening WhisperTest's compatibility across diverse app designs.

2.3 Related Work

Our library touches different domains such as mobile automation tooling, user-interface understanding, and mobile navigation agency. In this section, we review prior works in these domains.

2.3.1 Studies using mobile automation. Several recent studies have used mobile UI automation to analyze privacy and security behaviors of mobile apps at scale. DiOS [42] is one of the earliest systems

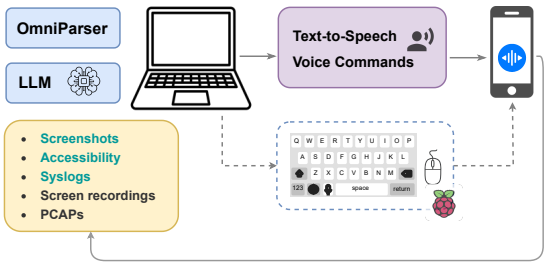


Figure 1: Overview of the WhisperTest automation pipeline. The controller machine optionally leverages LLMs and OmniParser to interpret UI content and issues voice commands via text-to-speech. Screenshots, accessibility data, syslogs are processed in real-time. Screen recordings and network traffic are saved for processing. Experimental keyboard, mouse and microphone emulation offers alternative input methods.

for dynamic privacy analysis of iOS apps, using Apple's UIAutomation framework on jailbroken devices. The study achieved limited code coverage and it was unable to handle login flows or other complex user inputs.

Kollnig et al. [10] performed a large-scale automated privacy analysis of iOS and Android apps, revealing widespread third-party tracking and sharing of unique identifiers on both platforms. They launched each app automatically on real jailbroken devices for network traffic analysis and captured data flows to tracking domains. However, they did not simulate any user interactions after app launch, citing the lack of established tools for automating and navigating arbitrary iOS app interfaces. This limitation likely caused the analysis to miss behaviors triggered only through user engagement.

Xiao et al. [43] automatically interacted with iOS apps using their tool LaLaine to assess whether declared App Store privacy labels were accurate, by dynamically executing the apps and analyzing whether they collected personal data inconsistent with their stated disclosures. However, the system faced limitations; approximately 20% of the apps could not be analyzed because they failed to run on jailbroken devices, and the automation framework sometimes failed to fully traverse app interfaces that required complex user inputs (e.g., login), a challenge we address using custom commands §4.2.

Koch et al. [4] conducted a large-scale analysis of dark patterns in privacy consent dialogs in Android and iOS apps. Using Appium, they identified consent dialogs via UI structure and keywords, then interacted by accepting or rejecting consent. The study faced notable limitations: Appium was unstable, occasionally crashing or failing to extract UI elements or screenshots, and on iOS, they were limited to an outdated iOS version due to the lack of a working jailbreak, leaving 9% of apps unanalyzed.

Mohamed et al. [15] analyzed Apple's App Tracking Transparency (ATT) prompts in iOS apps to assess their compliance and detect misleading language. They ran apps on jailbroken devices and interacted with the apps by dividing the screen into a 45-cell grid and randomly tapping cells for 30 seconds to trigger ATT or pre-permission dialogs. However, this straightforward method may

Table 1: Comparison of WhisperTest with existing iOS automation tools. "All" refers to the ability to automate Native, Hybrid, and Web apps. Tools such as `go-ios` and `libimobiledevice` are not included, as they offer similar device communication capabilities to `pymobiledevice3` but lack UI automation features. Supported Data: S = screenshots; P = PCAPs; L = syslogs; A = accessibility; V = video; O = OCR; U = UI tree; C = Crash reports.

Tool	3rd-Party Apps	App Types	Supported Data	macOS Only	Requires App Code	Language
WhisperTest	Yes	All	S, P, L, A, V, O	No	No	Python
Appium-XCUITest [3]	Yes	All	S, U, L, A	Yes	No	JS, Java, Python
EarlGrey [1]	No	Native, Hybrid	C, S, P, A	Yes	Yes	Obj-C, Swift
idb [22]	No	Native	C, L, A	Yes	No	Python
KIF [21]	No	Native	A	Yes	Yes	Obj-C
NoSmoke-iOS [33]	Yes	All	O	Yes	No	JS
`pymobiledevice3` [34]	Yes	All	S, P, L, A	No	No	Python

lead to unintended interactions and may not be able to handle more complex UI flows.

Reyes et al. [16] developed a dynamic analysis framework to evaluate COPPA compliance in Android children's apps, using Android's UI/Application Exerciser Monkey to simulate user interactions via pseudorandom input events. The study found widespread potential COPPA violations, often caused by misconfigured third-party SDKs. While the Monkey enabled scalable testing, its random interaction approach might have missed more complex, context-dependent app behaviors.

In another study focusing on Android apps for children, Zhao et al. [11] presented a framework to detect inappropriate ads. Its core component explores apps and interacts with in-app ads by leveraging Android Accessibility Services [44] to build a view tree and identify ad views based on their runtime class.

Seiden et al. [5] proposed a framework named AppTap for dynamically analyzing iOS apps by running `.ipa` files on Apple Silicon Macs, avoiding the need for jailbreaking. They analyzed how apps interact with system APIs and handle sensitive data during runtime. However, the approach is limited to macOS and suffered from a 30% app installation failure rate.

Tang et al. [7] developed a method to download and decrypt iOS app binaries, which they then analyze for vulnerable network services. They install, launch, and uninstall each iOS app using ideviceinstaller ([45]). However, their method relies on jailbroken devices and invoking an interface of the iTunes .dll files—a fragile approach, as minor changes by Apple can break compatibility.

2.3.2 Agent-Based Mobile Navigation. To improve mobile UI automation, prior work has explored processing user interfaces using screen pixels alone or in combination with view hierarchies, OCR, and metadata.[46–48]. OmniParser [49], which WhisperTest uses under the hood, combines icon detection and functional description models to convert UI screenshots into structured data. Li et al. [50] trained models to generate natural language descriptions for UI elements following a similar multimodal approach. Zhang et al. contributed Ferret-UI, an LMM developed to perform UI screen referring and grounding tasks. Yang et al. [51] introduced a visual prompting technique called Set-of-Mark to improve grounding in LMMs such as GPT-4V.

More recently, research is advancing from generating structured representations for mobile UI to using them as inputs for LLMs and Large Multimodal Models (LMMs) to generate entire interaction sequences [52–58]. For instance, Zhang et al. introduced AppAgent [59, 60], a multimodal framework combining structured grounding, reinforcement learning, and multimodal models for Android app navigation. AppAgent trains through exploration and human demonstrations, processing screenshots and XML UI data to select actions such as tap, swipe and text input.

Wen et al. [61] proposed AutoDroid, augmenting LLMs with memories of simulated app tasks and retrieving relevant past interactions by embedding task descriptions. Similarly, Wang et al. [62] proposed Mobile-Agent, which interprets screenshots, text, and icons via LLMs, and uses multiple agents to automate Android apps. Hong et al. [63] introduced CogAgent, which is a visual language model designed to enhance the understanding and navigation of both computer and smartphone GUIs.

2.3.3 Differences from the Related Work. While WhisperTest leverages LLMs for navigation, it does not aim to outperform existing models or benchmark their capabilities. Instead, our analyses serve to demonstrate the WhisperTest 's applicability, and its modular design allows integration with other systems for broader use cases. Lastly, while much prior work focuses on Android due to its openness, iOS remains underexplored—particularly in sensitive domains such as children's apps. To the best of our knowledge, no prior study analyzes third-party tracking and advertising specifically in iOS apps aimed at children. Similar efforts exist for Android [11, 16, 64] and the web [65], but not for Apple's ecosystem. WhisperTest helps bridge this gap and also enables new applications beyond security and privacy, which we outline in the following (§4.5).

Most prior studies rely on jailbroken iOS devices, limiting compatibility with recent OS versions and causing high app failure rates—up to 20% in some studies. In addition, several prior work using dynamic analysis have adopted naive UI interaction methods such as random tapping on the screen or not interacting with apps at all, limiting the effectiveness of dynamic analysis. Our study shows that WhisperTest can enable performing similar dynamic analysis research on a jailbreak-free setup.

3 WhisperTest: Design and Implementation

Developing automation tools for iOS is challenging due to its closed ecosystem and undocumented protocols such as lockdownd and RemoteXPC [66]. To simplify these complexities, WhisperTest leverages `pymobiledevice3` to abstract low-level communications, focusing instead on offering a reliable, lightweight interaction API similar to Selenium WebDriver.

WhisperTest operates without requiring app source code, jailbreaking, or macOS, enhancing its compatibility across platforms. Instead, our library relies on accessibility features that are guaranteed to remain available, ensuring long-term stability. Designed with extensibility in mind, WhisperTest is adaptable to a wide range of use cases.

3.1 Core Components

3.1.1 Controller Machine. As shown in Figure 1, WhisperTest runs on a controller machine that installs, launches, and interacts with iOS apps using voice commands or USB-based input emulation. During execution, WhisperTest continuously collects screenshots, screen recordings, syslogs, and network traffic—all of which are stored in the controller machine.

3.1.2 Accessibility Features and Voice Control. WhisperTest leverages the accessibility features of iOS, combined with Voice Control commands, to navigate through app interfaces and automate interactions. To monitor accessibility events, our library uses a separate thread to invoke the `AccessibilityAudit` service and receives the list of supported screen elements. For elements such as toggles the on/off state is included in the audits (Appendix A.9).

After scraping the accessibility details, WhisperTest can issue commands to tap an item or type in some text using Voice Control. The caller code that uses WhisperTest may choose the next action based on the app context and analysis goals. For instance, if the action is to tap an *"Accept"* button, WhisperTest will instruct the controller PC to play a voice command (*Tap accept*) and confirm that it is recognized by monitoring the device syslogs in real-time.

3.1.3 OCR and the Grid Method. Many apps lack accessibility labels, making UI elements invisible to accessibility audits. To overcome this, WhisperTest integrates OmniParser [49], a vision-based tool that uses OCR and icon detection to identify interactable UI components in screenshots. When accessibility data is unavailable, WhisperTest uses OmniParser to detect element types and positions using visual cues. WhisperTest then applies a grid-based method, dividing the screen into numbered rectangles (Figure 2) and interacting with the rectangle that matches the coordinates of the selected element.

3.1.4 Syslog Monitoring. In iOS, system logs (syslog) provide a comprehensive record of system events, errors, and diagnostic information generated by both the operating system and applications. Developers and forensic analysts utilize syslogs to monitor device behavior and troubleshoot errors. WhisperTest runs a separate thread to monitor syslogs, making them available through a queue. Our library then uses these logs to confirm voice command recognition, detect foreground app changes, and identify crashes. Additionally, WhisperTest allows for registering custom pattern matchers to receive any relevant logs.

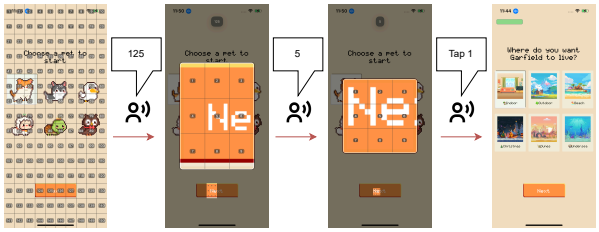


Figure 2: Handling inaccessible apps using the grid method.

3.1.5 Text-to-Speech (TTS). WhisperTest issues voice commands to the device using a local Text-to-Speech (TTS) module. To ensure offline execution and low latency, we use a locally deployed model from the Piper TTS project [67]. Piper converts text into speech without requiring access to cloud services. The TTS component supports multiple providers, and in this work, we used the English (US) "Amy" voice in medium quality from Piper's public voice library [68]. The controller generates audio files for each command, stores them locally, and plays them back. If a command fails or is not recognized, the system retries with configurable logic and verifies execution through syslog feedback. To evaluate the effectiveness of our TTS setup, we tested 40 commonly used commands from categories such as basic navigation, overlays, and gestures, achieving a recognition success rate of 99%. Additionally, we issued commands "Tap 1" through "Tap 100" in a silent environment to assess numerical recognition accuracy, obtaining a 98.6% recognition rate over five repetitions.

For loud environments, we developed an alternative, but experimental *silent* method to send the voice commands. Our method simulates a user talking to a USB microphone to control their phone. In particular, we used a Raspberry Pi (RPi) as a USB On-The-Go peripheral that emulates a microphone. WhisperTest sends the voice commands to RPi over HTTP, which then silently plays it to the test device.

3.1.6 USB Mouse and Keyboard Emulation. In addition to emulating a USB microphone, we emulate a USB mouse and keyboard connected to the iOS device using a Raspberry Pi. These emulated devices can be controlled programmatically by WhisperTest, providing an alternative method to control the iOS device in scenarios where voice input is undesirable or impractical. In this mode, the coordinates of the screen element that should be tapped are converted into mouse movements followed by a left click. Typed text is converted into the individual keystrokes a user would have to press in order to produce the string on a physical keyboard. Microphone, keyboard, and mouse emulation occupy the iPhone's only lightning or USB-C port, which WhisperTest also uses to receive syslogs, screenshots, and other data via `pymobiledevice3`. While `pymobiledevice3` supports connecting to devices over Wi-Fi, we did not use this configuration in our study.

3.1.7 PCAP Captures. To analyze network traffic generated by iOS applications, WhisperTest integrates a lightweight PCAP capturing module based on `pymobiledevice3`. WhisperTest starts the PCAP capture in a concurrent thread and accumulates the packets in a buffer. Upon stopping the capture, packets are written to a

standard PCAP file. While testing an app, WhisperTest supports starting and stopping packet captures multiple times. This enables a fine-grained analysis of the network activity triggered by specific UI interactions. While WhisperTest can capture network traffic, it cannot decrypt encrypted traffic such as TLS.

3.1.8 Custom Commands and Shortcuts. To extend the capabilities of WhisperTest beyond basic navigation, we leverage two native iOS features: Custom Commands and Shortcuts. iOS allows users to create custom voice control commands by assigning spoken phrases to specific gestures or sequences of actions. The commands can consist of taps, swipes, or launching an app. Custom Commands can be particularly effective for repetitive automation tasks, such as scrolling through social media feeds. We successfully used custom commands in our web measurement (§6) to scrape Safari's Privacy Reports and certificate details.

3.1.9 Screen Recording. WhisperTest uses iOS's Voice Control to initiate and stop screen recordings through custom commands, allowing the capture of app interactions without requiring elevated privileges. The video file is then copied to the controller machine.

3.2 Implementation and usage

To enable full functionality, users must install `pymobiledevice3`, mount the appropriate Developer Disk Image (DDI), and install `Piper` [67] and `playsound` [69] for generating and playing voice commands. The core `WhisperTestDevice` class handles app management, screen capture (via accessibility and OCR), PCAP recording, voice command execution, and screen recording (see Appendix A.3 for a usage example). Enabling Developer Mode on the test device is required for certain privileged operations, such as launching apps, while other features—accessibility-based screen scraping, PCAP capture, and syslog monitoring—function without it. On iOS 17 and above, a trusted tunnel must be established with `sudo` to interface with developer services.

4 Navigation Modules

Beyond the core components, we design specific navigation modules on top of our library.

4.1 Permission Dialog Handler

WhisperTest is capable of handling native iOS dialogs, such as the Apple Tracking Transparency (ATT) prompt, Location Services and Push Notifications. These dialogs are recognized through predefined heuristic rules applied to accessibility data. WhisperTest determines whether the dialog is present and selects the appropriate response based on user preferences (e.g., "Allow," "Don't Allow," or "Ask App Not to Track"). Once the intended action is selected, the command is issued through a voice command.

4.2 Apple Authenticator

We make use of Custom Commands to support specific tasks that benefit from consistent UI patterns or require repeated system-level actions. One key use case is automating Apple account authentication for apps that require login. Since the Apple login interface follows a relatively stable structure across apps, we define a custom voice command that reliably navigates this flow.

4.3 App Installer

Downloading and installing apps are the necessary first steps for large-scale iOS studies. Unlike Android, where `apk` files can be obtained from public registries or research datasets, each iOS app bundle is signed for the user's Apple ID, making it impossible to reuse apps downloaded by others. Tools such as IPATool allow for downloading and installing iOS apps for a given Apple account, but they rely on unofficial API endpoints. Case in point, during our study, downloading apps via IPATool became temporarily impossible for several months due to unannounced changes in Apple's infrastructure [37, 38]. To address this challenge, we developed a WhisperTest feature to download and install apps in bulk. Our method relies on a combination of voice commands and syslog monitoring to automate app installation. WhisperTest starts with opening a simple local web page with the App Store links of the apps to be installed. Using voice commands, each link is tapped, and WhisperTest then issues the command (`Get`, `Re-download`, or `Update`) depending on the app's prior installation status. Throughout the process, WhisperTest monitors syslog entries to confirm each step and ensure the app is successfully installed. In a preliminary test on a sample of 100 apps, WhisperTest successfully installed 98 of them, showing that the method works reliably across different common installation scenarios. The failures occurred because the system did not successfully activate the App Store link, preventing the installation process from starting. We are confident that the failed cases can be addressed with better error handling.

4.4 Consent Handler

One of the key use cases supported by WhisperTest is the automated handling of consent dialogs, which are commonly presented in mobile apps to comply with privacy regulations. WhisperTest addresses this in two stages: first, detecting if the current screen contains a consent prompt, and second, generating an interaction strategy based on the desired consent mode—accept or reject.

To detect consent dialogs, WhisperTest extracts screen content using accessibility audit data or OCR, and analyzes it with LLM. Navigation instructions are dynamically added to the LLM prompt if a consent dialog is detected. While accepting consent typically requires a single action, rejecting it is more complex, often involving additional steps such as navigating to settings and confirming choices—cases we address in our prompt design (see Appendix A.7 for details). Figure 3 illustrates how WhisperTest handles both native and cookie dialogs when operating in reject mode.

4.5 Other Use Cases

WhisperTest's ability to navigate apps while collecting a rich set of data makes it suitable for a range of research applications. In accessibility studies, it can evaluate whether apps expose proper metadata and support assistive technologies. In security, it helps detect TLS issues and audit local protocols, without needing to decrypt traffic. For privacy, it offers scalable analysis of third-party tracking and consent dialogs—an area underexplored on mobile. Legal and policy researchers can use it to assess GDPR compliance, detect dark patterns, and verify app store privacy claims.

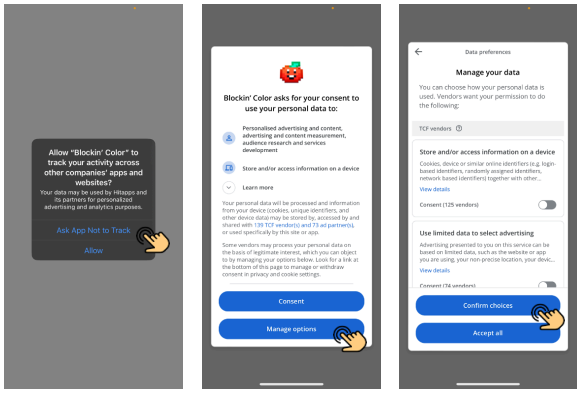


Figure 3: Example of handling native and consent dialog in reject mode. (The tapping hand icons highlight where interactions occur.)

5 Ads and tracking in children's apps

Recent studies highlight frequent tracking and advertising practices on apps and websites targeted at children—often violating privacy regulations [11, 16, 65, 70]. With mobile devices, particularly iPhones, becoming highly popular among younger users [71]. There is growing concern regarding children's exposure to harmful ads and tracking. While such issues have been studied on Android and the web, the iOS ecosystem remains underexplored, likely due to a lack of open-source automation tools. To fill this gap and demonstrate WhisperTest's capabilities, we analyze advertising and tracking practices of 200 iOS apps targeted to children. In particular, we compare the apps under two conditions: when users accept or reject tracking and personal data processing in GDPR consent dialogs, as well as in Apple's ATT prompts.

5.1 Navigation and Data Collection Pipeline

We developed an automated pipeline on top of WhisperTest that enables systematic installation, launching, navigation, and monitoring of mobile apps while collecting data related to advertising and privacy/security practices. The pipeline follows a hybrid approach, combining rule-based logic and LLM-driven decision-making. It interprets screen content using accessibility data, OmniParser outputs, or raw screenshots, and executes actions through voice commands.

The navigation process follows a tiered fallback strategy to maximize interaction success:

(1) Extract screen data using accessibility audits, which is a fast and lightweight method.
(2) If recognizable patterns (e.g., native iOS dialogs) are detected, apply predefined rules to interact.
(3) If the app is not accessibility-friendly or the first attempt fails, extract textual and structural information using OmniParser and apply rule-based heuristics on this output.
(4) If rule-based methods fail, prompt a vision-language model to interpret the screen image and suggest the next action.
(5) If the vision model returns an invalid command, fall back to a text-based LLM to interpret the screen—first using accessibility audit data, then OmniParser output.
(6) Once a valid interaction command is generated, execute it via voice command.

To verify successful screen transitions, WhisperTest monitors syslog messages to confirm that the device has recognized each voice command. Further, WhisperTest compares screenshots taken before and after each action using both visual and textual signals. It extracts OCR data via OmniParser and applies Jaccard similarity to detect content changes. Also, the average hash (aHash) from the `imagehash` library [72] as a lightweight and reasonably tolerant method is used to capture visual differences. A screen change is detected when both textual and visual differences are observed.

Additionally, we monitor syslog events to detect critical behaviors—e.g., if the app moves to the background, navigation is halted. The navigation phase continues for a fixed duration (200 seconds). Throughout this process, WhisperTest collects screenshots, accessibility audit data, OmniParser outputs, PCAP network traffic, and screen recordings. The recorded data is later analyzed to detect the presence of ads and third-party tracking.

Challenges in Navigating Children's Apps. Applications, and more specifically children's apps, often feature minimal textual content, relying heavily on vibrant, cartoonish imagery, making it difficult for traditional text-based parsers to detect key icons and buttons. For instance, we observed that critical icons, such as the play button, were frequently undetected by OmniParser due to their stylized, non-standard representations. Additional challenges included parental verification mechanisms such as age checks and mathematical puzzles, which must be completed to access settings, in-app purchases, or locked content. These mechanisms are likely implemented to restrict children's access to specific content or features and to ensure compliance with legal requirements such as the Children's Online Privacy Protection Act (COPPA) [73]. Subscription prompts presented directly at launch often require users to subscribe before proceeding. We dismissed these screens by locating subtle "Close" or "Continue with Limited Version" buttons (see Figure 6 in Appendix A.6). Multi-step login and registration flows further complicate interactions, as they demand an understanding of interaction sequences and not just individual actions per screen. To overcome these challenges, we employed a multimodal large language model instead of the simpler rule-based or template-matching methods. We leveraged `Qwen2.5-VL` [74], which is capable of localizing objects within images by generating bounding boxes or point coordinates. Also, we used a more efficient text-only model (`Qwen2.5-7B`) to detect the presence of consent dialogs and a larger model (`Qwen2.5-14B`) for text-based LLM navigation (in Appendix A.7 we explain how we chose these models). To address complex interaction flows, we developed a comprehensive set of navigation guidelines covering the scenarios observed in our pilot study. More importantly, we used the Chain-of-Thought (CoT) [75] prompting strategy—outlined under "Step-by-Step Reasoning" in Table 8—breaking down complex tasks into smaller actions. Additionally, providing in-context examples within the prompt [76] further enhanced the model's ability to cope with complex interfaces.

5.2 App Selection and Installation

To systematically analyze ads and tracking in children's apps on iOS, in a manual pilot study, we manually installed and reviewed the top 50 popular child-directed apps and found no visible ads during execution. Since our goal is to analyze tracking and ads, we refined our app selection strategy, in line with prior work [11], to ensure our dataset included apps likely to display ads. Specifically, we used the App Store Scraper [77], a Node.js module that extracts App Store metadata. Since the scraper does not support direct category-based searches for children's apps, we identified suitable child-related keywords using Google Keyword Planner [78], selecting and manually verifying the ten most popular and relevant keywords. The final list of keywords used for app discovery is provided in Appendix A.1. To further refine our dataset, we leveraged privacy metadata provided by the App Store Scraper to select apps that listed `Third-Party Advertising` as a declared purpose for data collection. We then applied additional filters to retain only those with a content rating of 4+, at least 5000 reviews, and a last update in or after 2023. After deduplication, this process yielded a final set of 313 apps. To validate this approach, we randomly selected 20 apps and confirmed that 13 contained ads during manual exploration.

To obtain the corresponding `.ipa` files, we used IPATool[79]. Out of the 313 apps, we successfully downloaded 217 `.ipa` files. All apps were downloaded from the App Store, with the region set to the Netherlands—the authors' country of residence. Download failures occurred for 94 apps: 21 apps were no longer available on the App Store, and 73 could not be downloaded, because a change introduced by Apple during our data collection broke IPATool's download functionality [38]. Note that these failures were due to limitations of IPATool and not related to WhisperTest, as also discussed in §4.3. While we considered using our app installer functionality (§4.3) to resume downloads, we opted to limit our experiment to random subset of 200 apps (of 217)—chosen as a practical round figure—to keep a consistent download method for all tested apps. Using IPATool also has the advantage of having access to `.ipa` files for static analysis. We used WhisperTest to systematically explore these apps, capturing data related to tracking and advertising and assessing the library's ability to automate real-world app interaction.

5.3 Experimental Setup

Figure 4 illustrates navigation in a cartoonish user interface of a children's app. Additional examples of different stages of WhisperTest's navigation pipeline are provided in Appendix A.6.

We tested WhisperTest across multiple devices and environments to ensure robustness and cross-platform compatibility. This included iPhone 13 devices (initially running iOS 17.5.1, later updated to 18.3.2), an iPhone 13 Pro with iOS 18.3.2, and an iPhone 8 running iOS 16.7.11. The library was evaluated on both macOS (MacBook Pro M1), Linux (Ubuntu 24.04.1 LTS), and Windows (11) machines. Across these platforms and iOS device models, WhisperTest successfully passed all tests and produced consistent results, with no differences in performance or outcomes. For the final data collection involving 200 apps, we used an iPhone 13 Pro (iOS 18.3.2) paired with a Linux laptop. Server-side LLM and vision model inference tasks were executed on Nvidia A100 GPUs (40GB VRAM), totaling approximately 20 GPU hours.

5.4 Third-party Tracker Detection

In this study, we make use of WhisperTest's automation features to detect the presence of third-party trackers in 200 iOS apps in two different scenarios: When the "Accept" option is selected in the consent dialog and when the "Reject" option is selected instead. First, WhisperTest is used to autonomously navigate each app while being instructed to either accept or reject any consent dialog that appears during navigation. The library automatically captures network traffic sent and received by the iOS device in PCAP format during this process. Although we are not able to decrypt encrypted traffic using this method, it allows us to extract the hostnames the apps try to connect to from DNS queries. However, DNS queries may provide an incomplete view due to local caching and DNS over HTTPS, the latter of which is sporadically used in our captures. Since this would render our detection unreliable, we additionally extract hostnames from the SNI (Server Name Indication) field of the TLS handshakes. All connections in our captures were found to support the SNI extension. We also do not find any Encrypted Client Hello messages in the app traffic, which would hide the actual SNI field from our analysis. We match the contacted hostnames to DuckDuckGo's Tracker Radar dataset [80]. Certain rules in the tracker dataset require a full request URL for tracker classification. This is because certain hostnames can be used to serve both benign and tracking-related content. Having only access to hostnames, we report tracker counts as a *range*: the minimums include hostnames classified as trackers without ambiguity, while the maximum also accounts for *potential* trackers inferred from known tracking-related *hostnames*.

5.5 Measurement Results

Out of the 200 apps (§ 5.2), WhisperTest successfully installed, launched, and navigated 195. Five apps failed to launch and crashed immediately upon execution. We reproduced these failures and examined the syslogs. The logs indicated that apps were terminated immediately after launch, but no specific cause was recorded. When we installed the same apps using WhisperTest 's App Installer (§4.3) or manually from the App Store, the apps launched and ran normally without crashing. This indicates that failures are likely due to IPATool, rather than WhisperTest. For each app, the complete experimental cycle—including installation, launch, navigation, and uninstallation—took approximately 329 seconds on average. The navigation phase lasted up to 200 seconds per app and was carried out using WhisperTest 's tiered interaction pipeline, which combines rule-based heuristics and LLM-driven decision-making. To better understand LLM usage during navigation, we logged all model queries issued throughout the data collection process. For simplicity, we analyze the data from the accept mode and observe a total of 823 LLM queries, averaging approximately four queries per app. Of these, 517 were directed to the vision-language model, with 336 resulting in successfully executed actions. The text-based LLM was queried 215 times using OCR (OmniParser) data as input, resulting in 147 successful interactions. When using accessibility data as input, the LLM was queried 91 times, with 46

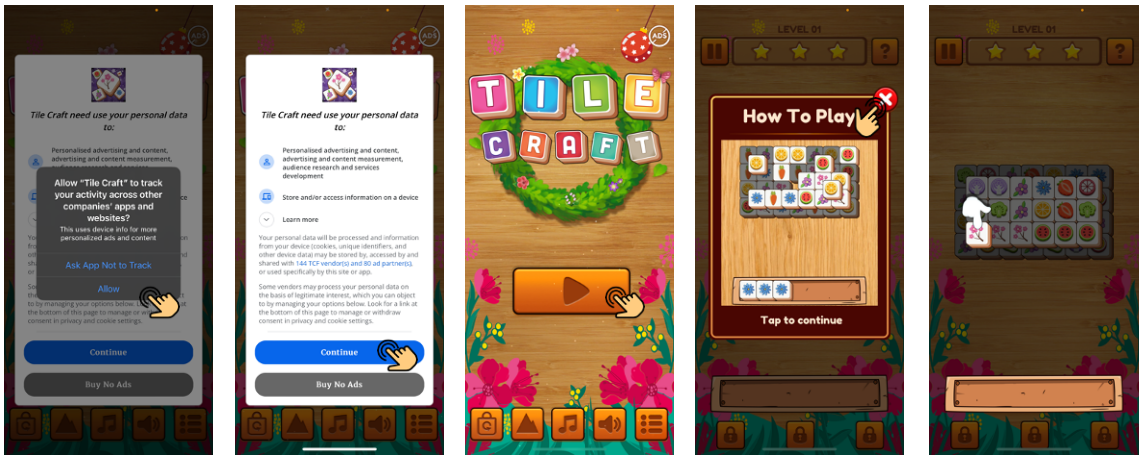


Figure 4: Example of navigating a cartoonish children's app interface, including interaction with a consent dialog and in-app elements required to proceed within the game.

successful responses. These numbers reflect the fallback structure of the navigation pipeline, where vision models are prioritized in the cartoonish UI environments typical of children's apps and text-based models serve as fallbacks when visual parsing fails. In some cases, the vision language model returned imprecise or misaligned coordinates, leading to failed interactions, highlighting the need for complementary text-based strategies to improve robustness. The average inference time for LLM responses during navigation was 5.84 seconds. WhisperTest used the custom voice command to authenticate with Apple (§4.2) in six apps where login prompts appeared during navigation.

Consent notices. WhisperTest detected and interacted with cookie consent dialogs in 89 of the 200 analyzed apps. Since many consent notices use deceptive design patterns to steer users toward accepting all cookies and data collection [81], we further analyzed how choices were presented. Only eight apps provided equally prominent accept and reject options on the first layer. In contrast, 73 lacked a visible reject option on the initial screen, and 67 offered none even in the second layer or settings. Additionally, 30 apps used color highlighting or other visual emphasis to draw attention to the accept button.

Third party trackers. Table 2 summarizes the number of third-party trackers observed under each consent mode. We find that apps contact only slightly fewer trackers in reject mode, indicating that users cannot rely on declining consent to avoid tracking. Table 3 further breaks down the trackers by category using DuckDuckGo's Tracker Radar dataset [80], revealing that advertising and analytics are the main purposes for tracking. When consent is declined, the number of contacted trackers is reduced roughly uniformly across all categories. Table 6 in the Appendix A.4 reveals that Google—represented by domains such as `doubleclick.net` (see Table 4), the most frequently observed tracker in our dataset—is dominant in the iOS ecosystem, appearing in over 67% of apps in accept mode, more than twice as prevalent as the next most common tracker, InMobi. We acknowledge that using contacted hostnames to measure tracking may be a crude approach. Since WhisperTest declines the ATT prompt in the reject mode, apps and trackers will receive a zeroed-out Identifier for Advertisers (IDFA), but they may still use signals such as IP address and device model, and they may assign a unique ID to track users within apps.

Table 2: Tracker domain counts measured across iOS apps while accepting and rejecting consent dialogs, respectively. Each column reports a range: lower bounds include only confirmed tracker domains, while upper bounds include additional potential trackers inferred from encrypted hostnames.

Metric	Accept	Reject
Total distinct tracker domains	512 - 943	409 - 805
Apps with at least one tracker	152 - 171	140 - 168
Mean tracker domains per app	2.63 - 4.84	2.11 - 4.15
Median tracker domains per app	3 - 5	2 - 4

Ad Statistics. We analyzed ads that appeared in both consent modes of children's apps. Using the template matching technique that we detail in Appendix A.2, we initially detected 634 candidate ads. After manual review to remove false positives (approximately 15%) and repetitions (the same ad appearing across multiple screenshots of a single app), we identified 131 unique ads in 50 apps under the accept mode, and 97 ads in 42 apps under the reject mode. The detected ads included promotions for high-risk financial products such as speculative trading platforms (e.g., Capital.com), which fall under *restricted categories* according to Google's ad policies and should not be shown to children [82]. We also observed retail promotions (e.g., Shein, Zalando) and interest-targeted content, such as an ad stating "Singles Should Read This", and a quiz-like ad asking "How rare is your intelligence type?". Notably, even at this small scale, we found clear ad policy violations, highlighting the need for broader analysis to assess inappropriate ads in child-directed iOS apps. Examples of detected ads are provided in Appendix A.5.

Table 3: Most common categories of third-party trackers across 200 iOS apps while accepting and rejecting consent dialogs, respectively. The right-hand columns show the total number of trackers in each category. Note that a single tracker may belong to multiple categories at once.

Category	Accept	Reject
Ad Motivated Tracking	131	120
Advertising	130	118
Analytics	94	83
Third-Party Analytics Marketing	70	60
Action Pixels	63	53
Audience Measurement	63	54
Ad Fraud	22	7
Embedded Content	5	5
Federated Login	2	0
Uncategorized	80	67

To assess the potential for missed ads, we manually reviewed a random sample of 20 apps from those where no ads were detected. We installed each app and interacted with it manually for 100 seconds. In seven out of 20 apps, we observed at least one ad, typically after bypassing challenging flows such as parental gates, age verification, or gameplay progression.

Unencrypted HTTP Connections. Analyzing the PCAPs captured during the analyses, we identified a total of 1,395 (unencrypted) HTTP requests, many of which did not redirect to HTTPS. Out of the 200 selected apps, 54 were found to make at least one HTTP request when accepting consent dialogs, compared to 44 when rejecting. While some were benign (e.g., 44 OCSP requests) others transmitted unique identifiers and downloaded game scripts and assets in plaintext. Notable examples include the following:

- A game made 860 separate HTTP requests to download game assets.
- An app sent obfuscated messages as well as unobfuscated tracking information to `startech.ltd`. The sent information included the device model and iOS version, country, and a unique identifier.
- Another app sent unique device identifiers, iOS version, country, and app interaction details to `api.adtrade.com`.
- A game downloaded a .zip archive of game scripts written in a Lisp dialect.
- Another game sent the user's IDFA in plaintext over an unencrypted WebSocket, along with device model and iOS version. Ad content was also retrieved without obfuscation.

While some observed requests were likely for connectivity tests, the use of HTTP to load scripts and send unique identifiers risks identity leakage and content tampering. App assets and ads are especially vulnerable, since their display is predictable. These concerns are not merely theoretical: threat actors have actively exploited unencrypted requests to redirect victims to malicious websites, deliver exploits, and ultimately compromise their devices [83].

Effect of Lockdown Mode. Given our discovery that several apps make insecure HTTP requests, we sought to understand whether such behavior persists under stricter security settings. In particular, we investigated Lockdown Mode [84], a feature introduced by Apple to mitigate the risk of sophisticated targeted attacks by significantly reducing the system's attack surface. This mode is intended for high-risk users and imposes a range of restrictions on app and browser behavior. Our focus was specifically on identifying insecure HTTP requests made by apps, traffic that is unencrypted and potentially vulnerable to interception and tampering. Surprisingly, we found that all three tested apps continued to make unencrypted HTTP (non-HTTPS) requests, even under Lockdown Mode. We reported this finding to Apple and recommended disabling insecure HTTP requests in Lockdown Mode, since such requests have previously been exploited to infect victims' phones with spyware [85].

Table 4: Most common tracker domains observed under both consent modes. The right-hand columns show the number of distinct apps where at least one tracker associated with the respective domain was detected.

Tracker Domain	Accept	Reject
doubleclick.net	123	113
googlesyndication.com	78	79
inmobi.com	62	50
googletagservices.com	60	51
amazon-adsystem.com	38	30
appsflyer.com	33	30
amplitude.com	22	21
doubleverify.com	15	0
2mdn.net	12	2
smaato.net	9	5

6 Using WhisperTest for Web Measurements

Web automation libraries such as Selenium [6] and Playwright [86] enabled numerous web security and privacy research [87–89]. These libraries also provided the foundation for web privacy measurement tools such as OpenWPM [90]. On iOS, web content inspection and browser automation can be performed using the driver object in `pymobiledevice3`, which leverages iOS' WebInspector service [91]. However, WebInspector can only inspect and automate the web content, not the browser's *chrome* [92] interface, including menus and toolbars. In this illustrative exercise, we show how WhisperTest, powered with custom commands, can be used for web measurements. Specifically, we scrape Safari's Privacy Report [93] and the Connection Security Details pages [94], which show the detected trackers and TLS certificate details, respectively. We define custom commands to open the two pages, and use accessibility audits to scrape their contents. OCR or LLMs were not needed for this web measurement. To avoid errors, custom commands were only issued when Privacy Report and Connection Security Details pages were available—we detect this by scraping Safari's *Page Menu*.

Before the crawl, we enabled iPhone's Lockdown mode [84], which limits certain functionalities, including web fonts and WebRTC [95]. This limited exercise aims to identify if the Lockdown mode causes widespread breakage, which may curtail its adoption. We choose target websites from Hanley and Durumeric's 2024 study

on detecting synthetic news articles [96]. They compile a list of news websites from prior work, categorized as *reliable* (mainstream news) or *unreliable* (misinformation, disinformation, propaganda). In particular, we compare 50 reliable websites to 50 unreliable websites regarding tracker prevalence and certificate authorities. Our measurements are limited in scale, as the WebInspector service frequently becomes unresponsive after visiting a few dozen websites—a problem commonly reported by other users [30]. Since our primary goal is to showcase WhisperTest's capabilities, versatility, and ease of use—rather than exhaustive measurements—we believe a smaller-scale study is sufficient. Moreover, the observed hangs can be addressed through improved error handling and recovery, rather than fundamental limitations of WhisperTest itself. Finally, opening websites is also possible through Shortcuts and custom commands, both of which can be invoked by WhisperTest and do not rely on the WebInspector service.

After loading a website, WhisperTest executes custom voice commands to open Safari's Privacy Report and the Connection Security Details page. We scrape their text through accessibility audits, capturing details about trackers and the website's certificate. Additionally, we collect the page source, page load time, document title, and innerText— the latter being used to identify page load errors. Screenshots of Safari's Page Menu and the homepage provide additional evidence (Figure 9 in Appendix A.8). The crawler does not interact with consent dialogs; thus, our measurements capture the tracking that occurs before user consent. In fact, during an automated session, Safari does not allow interaction with the page content, but we observed that certain voice commands (e.g., *scroll down*) bypass that limitation. Crawls from a European IP in April 2025 produced 80 successful visits (40 reliable, 40 unreliable), after excluding failures from unreachable (9) or region-locked (5) sites.

6.1 Results

Tracker Entities. Safari's Privacy report lists the detected tracker domains' owner entities (or companies). Google was the top entity in both website types with 78% prevalence on both (Table 5). Notably, advertising companies Adscore and MGID only appeared on unreliable websites, albeit infrequently (5 and 3 websites, respectively). In a recent study, Adscore was the most prevalent fingerprinter on the web [97]. Prior research on clickbait and deceptive ads found that MGID displayed ads on misinformation websites [98].

Tracker Domains. We found that reliable websites contain on average 9.4 tracker domains, as determined by Safari's Privacy Report, while unreliable ones had only 4.4, based on Safari's Privacy Report. The difference (p=0.002) may reflect differences in business models, popularity, or advertisers avoiding misinformation websites. For instance, doubleclick.net, Google's domain associated with advertising, was present on 70% of the reliable websites, but only 25% of the unreliable websites. On the other hand, google-analytics.com has a very similar prevalence on reliable and unreliable websites: 60% and 55%, respectively (Table 9, Appendix A.8).

Certificate Authorities. Combined, Google and Let's Encrypt issued 88% of all leaf certificates observed in our 80-website sample (G: 38, LE: 32). Comparing across reliable and unreliable websites, we find that Google's WE authority is used on 64% of the unreliable websites, but only 33% of the reliable websites. Reliable websites used Let's Encrypt more often than their unreliable counterparts (47% vs 31%), which may be due to limited technical capabilities.

Table 5: Most prevalent tracking entities found in the web measurement study on reliable and unreliable websites in Lockdown mode. Based on 40 websites in each category.

Entity (Reliable sites)	Num. sites	Entity (Unreliable sites)	Num. sites
Google	31	Google	31
Cloudflare	12	Adscore	5
Criteo	10	OneSignal	5
Amazon.com	10	Facebook	4
PubMatic	9	Prospect One	4
Facebook	9	PayPal	3
ID5	8	MGID	3
The Trade Desk	8	Amazon.com	3
WarnerMedia	7	Twitter	3
Taboola	7	Intuit	2

Breakage due to Lockdown Mode. We do not observe widespread breakage in our limited sample. The only issue was broken icons on 14 websites that identified by inspecting the homepage screenshots of 80 websites. Manual investigation of the page elements revealed that breakages stem from font-awesome [99] and similar libraries that use web fonts to render icons. Since Lockdown mode disables web fonts [84], icons appear as an empty rectangle. We visited the same websites on an iPhone and verified that the icons appeared after disabling the Lockdown mode. The missing icons do not pose a serious issue when accompanied by text. However, we find cases where icon-only menu designs may lead to confusion or unintended clicks, since several menu items are represented by the same empty rectangles (Figure 10 in Appendix A.8).

7 Safety and Security

As with all automation libraries, WhisperTest can be used to perform undesired activities and may carry safety and security risks. This section outlines key concerns and strategies to mitigate them.

Anti-debugging. While apps under test cannot directly detect if Developer Mode or Voice Control is enabled [100], there may be side channels to detect if Voice Control is enabled. When apps detect they are tested, they may behave differently, leading to results that are not representative of users' experience at large.

Malicious Bots. WhisperTest can be used to perform fraudulent activities such as fake interactions, sending spam comments, and unauthorized data scraping. By relying on Voice Control, WhisperTest has an inherent friction that rate-limits excessive activity, enabling scaling up of malicious activities. Each interaction takes several seconds to be sent and executed, internally limiting WhisperTest's speed. Regardless, WhisperTest users are advised to implement necessary safeguards such as sensible delays between actions and maximum retry limits to avoid unintended outcomes.

Tested Apps as Adversaries. Automating arbitrary apps may expose untrusted input. Malicious buttons or links could trigger unwanted actions or capture the test persona's credentials. Accessibility labels used by WhisperTest can also be manipulated

to conceal attacks. For scenarios involving credentials or sensitive data, mitigations include disabling certain Voice Control commands or restricting them to specific apps. Detailed system logs from the device can be used to verify iOS platform's own dialogs. For instance, App Store's app installation dialog is indicated by a log entry from "*appstored(AppleMediaServices)...*". Similarly, the *Sign in with Apple dialog* can be detected by logs similar to *"<Notice>: com.apple.AuthKitUIService: Foreground: true"*. WhisperTest users should be very specific when searching for expected strings in syslogs, to prevent against apps that may potentially inject arbitrary strings into syslogs.

Prompt Hijacking. Since WhisperTest relies on voice commands for app navigation, it is susceptible to prompt hijacking, where an app intentionally misleads the automation system by altering button labels or introducing carefully crafted UI elements. Similar injection attacks exist against Web UI automation systems that may lead to personal identifiable information exfiltration [101]. Hidden voice commands can be mitigated by muting the device. WhisperTest users may add a malicious prompt detection layer before sending UI labels to the models for next action prediction.

Elevated Privileges. The `pymobiledevice3` generally works without root privileges. For high-risk actions such as app installation, however, it must be started with root in a separate terminal to create a kernel virtual network device (TUN). While sudo carries security risks, WhisperTest itself never runs with elevated privileges. An open feature request for `pymobiledevice3` suggests a sudo-less alternative for creating a trusted tunnel is possible [102].

Information Leaks. Using external language, vision, or OCR models (e.g., OpenAI) may transfer sensitive on-screen data. To mitigate this, we prefer local LLMs over proprietary API-based models. WhisperTest is intended for test devices and simulated personas, and users should be cautious when applying it to apps with real personal data. We will also include clear safety and security warnings on WhisperTest's website and source repository. [2].

8 Discussion

Analyzing Broader App Categories. Our experiments focused on children's apps, which often feature highly stylized, non-standard UIs that challenge traditional UI testing. To investigate WhisperTest's applicability beyond children's apps, we conducted a small-scale analysis of non-children's apps. We randomly selected 50 popular non-children's apps from the App Store's top 100 free apps in the Netherlands, which included apps from categories such as social media and communication (e.g., Instagram, Signal, X), travel and transportation (e.g., Airbnb, Waze, Maps), shopping (e.g., Amazon, Temu), utilities (e.g., Outlook, Translate), entertainment (e.g., Spotify, Netflix), and education (e.g., Duolingo). WhisperTest successfully installed, launched, and navigated all 50 apps without error, based on the review of syslogs and screen captures. On average, it reached 5.29 unique screens and triggered connections to 29.48 distinct IP addresses per app. During testing, apps produced a median of 39.5 TCP/UDP flows and 1.48 MiB of network traffic, indicating active execution. For 13 apps, WhisperTest used its Apple Authenticator feature (§4.2) to log in. In five of those cases, the app required filling additional forms, which could not be completed—a known challenge for automated testing. Although customizing prompts could have partly addressed the form completion issue, we used our original prompts from §5 for consistency. This preliminary analysis suggests WhisperTest is promising for automating and analyzing apps from broader (non-children) categories. Yet, we acknowledge that our sample size is limited given the sheer scale and diversity of iOS apps that exist in the wild.

Comparison with Prior Research. Section 2.1 and Table 1 compared available tools (e.g., Appium), and §2.3.1 contrasted our work with prior research using mobile automation. Here, we add a comparison to related studies, focusing on tested apps, coverage, reliability, and other limitations or strengths. Prior work differs in both the number of apps tested and the automation strategies used. A 2024 study by Mohamed et al. [15] employed a random click approach to trigger ATT alerts on 4,680 apps. Xiao et al. used LaLaine [43] to analyze 6,332 apps using a depth-first search algorithm with a maximum depth of five UI views. DiOS [42] was used to test 1,136 apps with three execution types, including a smart strategy that keeps track of a state model graph of already executed UI paths. In contrast, WhisperTest 's fallback-based interaction pipeline we used in the study is designed for deeper, more realistic exploration. On the other hand, WhisperTest should also be viewed as a neutral automation library akin to web automation libraries such as Selenium or Playwright. While we use a specific pipeline for this study, different automation strategies can be built on top of WhisperTest, from random clicking to depth- or breadth-first search. We used WhisperTest to test 200 apps to analyze tracking and ads in children's apps, whereas studies by Kollnig et al.[10], Zhao et al.[11], and Binns (static analysis) et al. [64] tested 24,000, 25,000, and 959,426 apps, respectively. While our experiments are limited in scale our core contribution lies in developing the WhisperTest. The accompanying experiments serve as an exploratory analysis, designed to demonstrate WhisperTest's potential and robustness under challenging test conditions. A broader evaluation across diverse application categories is beyond the scope of this paper and is left for future work.

Regarding coverage, while WhisperTest does not support traditional code coverage analysis—a limitation we discuss in the next section, we provide a practical comparison with prior work by examining the number of tracker domains (eTLD+1) detected during app execution. In our study, the median tracker count was 3–5 in accept mode and 2–4 in reject mode as reported in Table 2. Kollnig et al. report a median of two tracker domains included in an app on both Android and iOS, while Binns et al. report a median of ten tracker domains. Note that the latter figures are derived from static analysis, where APKs were downloaded and unpacked using APKTool to identify host references, capturing all potential tracker endpoints, not just those actually contacted during execution.

Among most tools, failure to analyze or execute some apps remains a common issue: LaLaine could not analyze 1,200+ apps (19.4%) due to jailbreak issues. The random click approach skipped 188 apps needing a newer iOS version, and reached target dialogs in only 60.6% (2,836) of apps. AppTap [5] avoided device modification

[2] A potential warning message could be the following: *WhisperTest is an experimental tool and must be used with extreme caution. Its use may lead to data theft, data loss, or device compromise when used with untrusted apps or websites. When used with LLMs WhisperTest may be vulnerable to prompt injection attacks, which can result in unauthorized and harmful actions. WhisperTest must only be used on test devices with simulated data. Use on devices with real user data and accounts is strictly discouraged and may result in irreversible consequences.*

by running iOS apps on macOS, but faced compatibility issues, executing only 92 of 200 apps (46%). Zhao et al. [11] could only collect ads from 8,971 of 25,000 apps, mainly due to app crashes or apps requiring complex human interactions to display ads. WhisperTest failed to launch 5 of 200 children's apps due to IPAtool-related issues, but succeeded using its own launcher. All 50 non-children's apps were launched and ran without error. Moreover, while prior work [4] noted Appium's instability, WhisperTest ran reliably for over 20 hours without crashes and other fatal errors.

Our results should be seen as lower bounds, as ads, trackers, or privacy issues may be missed in apps requiring complex interactions such as gameplay, registration, or parental gates. Prior work found ads in 36% of children's (Android) apps labeled "Contains ads" [11], while we observed 25% in Accept mode—a comparable portion given the tested apps are from different platforms. This common shortcoming of ad automation could be reduced by task-specific scripts and more refined LLM prompts in future work.

Differences Across Usage Modes. WhisperTest supports multiple usage modes, including configurable consent handling (Accept/Reject) and testing under Apple's Lockdown Mode. While WhisperTest could reliably reject consent dialogs despite deceptive, multi-step designs, this only reduced ads and trackers to a limited extent. The number of observed ads dropped from 131 (median 3-5 trackers per app) in accept mode to 97 (median 2-4 trackers per app) in reject mode.

We also assessed system-level protections by enabling Lockdown Mode. Despite its stricter constraints, apps continued to transmit unencrypted HTTP traffic, indicating that Lockdown Mode does not fully mitigate insecure network behavior.

8.1 Limitations

A key limitation lies in WhisperTest's reliance on textual and visual signals for screen understanding, which might be incomplete or missing. Some apps lack accessibility support, and OmniParser captures only visible on-screen content, overlooking off-screen elements and sometimes missing icons. Unlike Android, iOS does not expose the full UI tree on non-jailbroken devices, limiting deeper UI analysis. While WhisperTest leverages syslog monitoring to confirm voice command execution or detect backgrounding, visibility of the app's internal state remains constrained. As a result, WhisperTest cannot perform traditional coverage analysis or reliably measure navigation depth. Instead, it uses a heuristic method that compares on-screen text and visual similarity between consecutive screenshots to infer screen transitions. While this offers a practical, non-privileged solution, it has inherent limitations: minor UI animations may trigger false positives, while subtle or non-visual transitions may go undetected. Future work could explore richer use of syslog data to improve state detection.

WhisperTest relies on voice commands, which may be missed or misinterpreted in noisy environments. To mitigate this, we can emulate a USB microphone for direct audio injection (§ 3.1.6), though this was not used in our study.

WhisperTest can capture network traffic but cannot decrypt HTTPS payloads. While this limits access to full request and response content; metadata such as queried domains, connection timing, and SNI fields still provide valuable signals for privacy and security analysis. Moreover, captures contain local communication protocols such as ARP and mDNS, enabling studies into the security of such protocols [103]. These protocols could not be captured using a VPN or proxy. Importantly, all measurements are conducted on non-jailbroken devices, which reflects a realistic user setup and avoids side effects introduced by jailbreak-based tools. However, WhisperTest is compatible with jailbroken devices as well.

As described in §5.1, we designed a layered strategy to navigate within children's apps to surface ads and third-party trackers. However, as our primary focus is demonstrating the broad applicability of WhisperTest, we did not exhaustively evaluate our navigation performance using standard mobile navigation benchmarks [104, 105]. Similarly, while we considered comparing WhisperTest to existing alternatives in a representative benchmark, the lack of cross-platform alternative prevented meaningful evaluation.

Our app analysis and web measurement were limited to 250 apps and 100 websites, respectively. While the sample size limits generalization, it demonstrates WhisperTest's capabilities and its potential in larger-scale privacy and security studies. The limited scale partly stems from the inherent slowness of using voice commands—a key safety feature limiting misuse (§7). In addition, we found the WebInspector to be less reliable than the rest of the `pymobiledevice3` library, which constrained the extent of our web measurement. However, these issues can be mitigated by careful monitoring and recovery, which we left out of scope.

Finally, our app selection relied on Apple's 4+ age rating and App Store privacy labels indicating third-party advertising. While practical, this may have included general-purpose apps or excluded others due to incomplete disclosures. We adopted this strategy after finding no ads in a pilot test of the top 50 ranked apps. Although this filtering may bias the sample toward ad-based monetization, it aligns with prior work [11]. We acknowledge this trade-off and that our findings may not generalize to all children's apps.

9 Conclusion

We presented WhisperTest, a new cross-platform, extensible, open-source library that enables iOS UI automation without jailbreaking. WhisperTest integrates voice commands with low-level system monitoring through syslogs and PCAP captures, and can be extended with different automated navigation approaches such as rule-based methods, template matching, and multimodal LLM-based methods to handle complex workflows. To validate WhisperTest's modularity and versatility, we performed various case studies, including an analysis of tracking and ads in apps targeted to children using multimodal LLM-guided automation. Our findings show that declining GDPR consent dialogs have a limited effect on tracking, and the use of unencrypted connections still persists at a non-negligible scale. Through a small-scale web privacy measurement exercise, we find differences in third parties used on mainstream and disinformation websites. Future work could integrate alternative GUI agent approaches to enhance coverage and apply WhisperTest across diverse research domains that can benefit from open, cross-platform and extensible mobile automation.

Acknowledgments

We would like to thank our reviewers for their valuable inputs to improve our paper. Gunes Acar is supported by a Netherlands Organisation for Scientific Research (NWO) Vidi grant. This work has also been partially supported by the Government of Canada's New Frontiers in Research Fund (NFRF), NFRFE-2019-00806.

References

[1] Google. EarlGrey: iOS UI Automation Test Framework. https://github.com/google/EarlGrey, 2024. Accessed: April 1, 2025.

[2] Apple Inc. XCUIAutomation | Apple Developer Documentation. https://developer.apple.com/documentation/xcuiautomation, April 2025. [Online; accessed 13-Apr-2025].

[3] Appium Contributors. Appium XCUITest Driver Documentation. https://appium.github.io/appium-xcuitest-driver/latest/, 2024. Accessed: 2025-04-02.

[4] Simon Koch, Benjamin Altpeter, and Martin Johns. The {OK} is not enough: A large scale study of consent dialogs in smartphone applications. In *Procs. of USENIX Security Symposium*, 2023.

[5] Steven Seiden, Andrew M Webb, and Ibrahim Baggili. Tapping. ipas: An automated analysis of iphone applications using apple silicon macs. *Forensic Science International: Digital Investigation*, 52:301871, 2025.

[6] Selenium. https://www.selenium.dev, April 2025. [Online; accessed 12. Apr. 2025].

[7] Zhushou Tang, Ke Tang, Minhui Xue, Yuan Tian, Sen Chen, Muhammad Ikram, Tielei Wang, and Haojin Zhu. {iOS}, your {OS}, everybody's {OS}: Vetting and analyzing network services of {iOS} applications. In *Procs. of the USENIX Security Symposium*, 2020.

[8] Hashida Haidros Rahima Manzil et al. Dynamaldroid: Dynamic analysis-based detection framework for android malware using machine learning techniques. In *International Conference on Knowledge Engineering and Communication Systems (ICKES)*. IEEE, 2022.

[9] Priyanka Bose, Dipanjan Das, Saastha Vasan, Sebastiano Mariani, Ilya Grishchenko, Andrea Continella, Antonio Bianchi, Christopher Kruegel, and Giovanni Vigna. Columbus: Android app testing through systematic callback exploration. In *Procs. of the International Conference on Software Engineering (ICSE)*, May 2023.

[10] Konrad Kollnig, Anastasia Shuba, Reuben Binns, Max Van Kleek, and Nigel Shadbolt. Are iphones really better for privacy? a comparative study of ios and android apps. *Proceedings on Privacy Enhancing Technologies*, (2):6–24, 2022.

[11] Yanjie Zhao, Tianming Liu, Haoyu Wang, Yepang Liu, John Grundy, and Li Li. Are mobile advertisements in compliance with app's age group? In *Procs. of the ACM Web Conference*, 2023.

[12] Shunguo Yan and PG Ramachandran. The current status of accessibility in mobile apps. *ACM Transactions on Accessible Computing (TACCESS)*, 12(1):1–31, 2019.

[13] Android Developers. Android Debug Bridge (adb). https://developer.android.com/tools/adb, 2024. Accessed: 2025-03-29.

[14] Appium Contributors. Appium documentation. https://appium.io/docs/en/latest/, 2024. Accessed: 2025-03-29.

[15] Reham Mohamed, Arjun Arunasalam, Habiba Farrukh, Jason Tong, Antonio Bianchi, and Z Berkay Celik. {ATTention} please! an investigation of the app tracking transparency permission. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 5017–5034, 2024.

[16] Irwin Reyes, Primal Wijesekera, Joel Reardon, Amit Elazari Bar On, Abbas Razaghpanah, Narseo Vallina-Rodriguez, Serge Egelman, et al. "won't somebody think of the children?" examining coppa compliance at scale. In *The 18th Privacy Enhancing Technologies Symposium (PETS 2018)*, 2018.

[17] Ailton Santos Filho, Ricardo J Rodríguez, and Eduardo L Feitosa. Evasion and countermeasures techniques to detect dynamic binary instrumentation frameworks. *Digital Threats: Research and Practice (DTRAP)*, 3(2):1–28, 2022.

[18] Apktool | Apktool. https://apktool.org, April 2025. [Online; accessed 13-Apr-2025].

[19] SensePost. objection. https://github.com/sensepost/objection, April 2025. [Online; accessed 13-Apr-2025].

[20] Yuvraj Agarwal and Malcolm Hall. Protectmyprivacy: detecting and mitigating privacy leaks on ios devices using crowdsourcing. In *Proceeding of the annual international conference on Mobile systems, applications, and services*, 2013.

[21] KIF Contributors. Kif - keep it functional. https://github.com/kif-framework/KIF, 2024. Accessed: April 1, 2025.

[22] Facebook. idb: ios development bridge. https://github.com/facebook/idb, 2025. Accessed: March 30, 2025.

[23] Xamarin. Calabash - automated ui acceptance testing for mobile apps. https://github.com/calabash/calabash-ios. Archived, last updated in 2017. Accessed: April 2, 2025.

[24] Sundos Mojahed, Réjean Drouin, and Lokman Sboui. Odace: An appium-based testing automation platform for android mobile devices certification. In *Procs. of IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2024.

[25] Junmei Wang and Jihong Wu. Research on mobile application automation testing technology based on appium. In *2019 International Conference on Virtual Reality and Intelligent Systems (ICVRIS)*, pages 247–250, 2019.

[26] Ashwaq A Alotaibi and Rizwan J Qureshi. Novel framework for automation testing of mobile applications using appium. *International Journal of Modern Education and Computer Science*, 9(2):34, 2017.

[27] Gabriel Lovreto, Andre T. Endo, Paulo Nardi, and Vinicius H. S. Durelli. Automated tests for mobile games: An experience report. In *2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2018.

[28] Denis Vajak, Ratko Grbić, Mario Vranješ, and Dejan Stefanović. Environment for automated functional testing of mobile applications. In *2018 International Conference on Smart Systems and Technologies (SST)*, 2018.

[29] WebDriverAgent - Appium XCUITest Driver. https://appium.github.io/appium-xcuitest-driver/4.16/wda-custom-server. [Online; accessed 13-Apr-2025].

[30] appium-xcuitest-driver/docs/guides/troubleshooting.md at master · appium/appium-xcuitest-driver. [Online; accessed 14. Apr. 2025].

[31] iOS tests suddenly became noticeably slower - Issues/Bugs - Appium Discuss, November 2021. [Online; accessed 14. Apr. 2025].

[32] Raiyan Rahman Chowdhury, Syeda Sumbul Hossain, Yeasir Arafat, and Bushrat Jahan Siddiqui. Configuring appium for ios applications and test automation in multiple devices. In *Procs. of the 2020 Asia Service Sciences and Software Engineering Conference*, 2020.

[33] Macaca Team. Nosmoke: Ui automation framework for mobile apps. https://github.com/macacajs/NoSmoke, 2021. Accessed: April 8, 2025.

[34] pymobiledevice3. https://github.com/doronz88/pymobiledevice3, August 2024. [Online; accessed 7. Aug. 2024].

[35] Daniel Paulus. go-ios: Golang based ios interaction tools. https://github.com/danielpaulus/go-ios, 2024. Accessed: April 8, 2025.

[36] libimobiledevice Team. libimobiledevice: A cross-platform protocol library to communicate with ios devices. https://libimobiledevice.org, 2024. Accessed: April 8, 2025.

[37] Majd Taby. ipatool issue #284. https://github.com/majd/ipatool/issues/284, 2024. Accessed: Mar 16, 2025.

[38] Majd Taby. ipatool issue #357. https://github.com/majd/ipatool/issues/357, 2024. Accessed: Mar 25, 2025.

[39] USB on the Go and Embedded Host | USB-IF. https://www.usb.org/usb-on-the-go, April 2025. [Online; accessed 13-Apr-2025].

[40] pymobiledevice3 developer accessibility crash · Issue #405 · doronz88/pymobiledevice3. https://github.com/doronz88/pymobiledevice3/issues/405#issuecomment-1465104729. [Online; accessed 13. Apr. 2025].

[41] Apple Inc. Use voice control on your iphone, ipad, or ipod touch, 2024. Accessed: April 9, 2025.

[42] Andreas Kurtz, Andreas Weinlein, and Christoph Settgast. Dios: Dynamic privacy analysis of ios applications. Technical report, 2014.

[43] Yue Xiao, Zhengyi Li, Yue Qin, Xiaolong Bai, Jiale Guan, Xiaojing Liao, and Luyi Xing. Lalaine: Measuring and characterizing {Non-Compliance} of apple privacy labels. In *Procs. of the USENIX Security Symposium*, 2023.

[44] Android Developers. Accessibilityservice. https://developer.android.com/reference/android/accessibilityservice/AccessibilityService, 2025.

[45] libimobiledevice.org. ideviceinstaller. https://github.com/libimobiledevice/ideviceinstaller, 2020. Accessed: March 28, 2025.

[46] Shuai Wang, Weiwen Liu, Jingxuan Chen, Yuqi Zhou, Weinan Gan, Xingshan Zeng, Yuhan Che, Shuai Yu, Xinlong Hao, Kun Shao, Bin Wang, Chuhan Wu, Yasheng Wang, Ruiming Tang, and Jianye Hao. Gui agents with foundation models: A comprehensive survey, 2025.

[47] Huawen Shen, Chang Liu, Gengluo Li, Xinlong Wang, Yu Zhou, Can Ma, and Xiangyang Ji. Falcon-ui: Understanding gui before following user instructions, 2024.

[48] Gilles Baechler, Srinivas Sunkara, Maria Wang, Fedir Zubach, Hassan Mansoor, Vincent Etter, Victor Cărbune, Jason Lin, Jindong Chen, and Abhanshu Sharma. Screenai: A vision-language model for ui and infographics understanding, 2024.

[49] Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. Omniparser for pure vision based gui agent, 2024.

[50] Yang Li, Gang Li, Luheng He, Jingjie Zheng, Hong Li, and Zhiwei Guan. Widget captioning: Generating natural language description for mobile user interface elements, 2020.

[51] Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v, 2023.

[52] Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. Large language model-brained gui agents: A survey, 2025.

[53] Zichen Zhu, Hao Tang, Yansi Li, Dingye Liu, Hongshen Xu, Kunyao Lan, Danyang Zhang, Yixuan Jiang, Hao Zhou, Chenrun Wang, Situo Zhang, Liangtai Sun, Yixiao Wang, Yuheng Sun, Lu Chen, and Kai Yu. Moba: Multifaceted

memory-enhanced adaptive planning for efficient mobile task automation, 2025.
[54] Jiayi Zhang, Chuang Zhao, Yihan Zhao, Zhaoyang Yu, Ming He, and Jianping Fan. Mobileexperts: A dynamic tool-enabled agent team in mobile devices, 2024.
[55] Filippos Christianos, Georgios Papoudakis, Thomas Coste, Jianye Hao, Jun Wang, and Kun Shao. Lightweight neural app control, 2025.
[56] Songqin Nong, Jiali Zhu, Rui Wu, Jiongchao Jin, Shuo Shan, Xiutian Huang, and Wenhao Xu. Mobileflow: A multimodal llm for mobile gui agent, 2024.
[57] Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework, 2024.
[58] Xiaoqiang Wang and Bang Liu. Oscar: Operating system control via state-aware reasoning and re-planning, 2024.
[59] Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users, 2023.
[60] Yanda Li, Chi Zhang, Wanqi Yang, Bin Fu, Pei Cheng, Xin Chen, Ling Chen, and Yunchao Wei. Appagent v2: Advanced agent for flexible mobile interactions, 2024.
[61] Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. Autodroid: Llm-powered task automation in android. In *Procs. of the 30th Annual International Conference on Mobile Computing and Networking*, ACM MobiCom '24, New York, NY, USA, 2024. Association for Computing Machinery.
[62] Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration, 2024.
[63] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. Cogagent: A visual language model for gui agents. In *Procs. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2024.
[64] Reuben Binns, Ulrik Lyngs, Max Van Kleek, Jun Zhao, Timothy Libert, and Nigel Shadbolt. Third party tracking in the mobile ecosystem. In *Procs. of the 10th ACM Conference on Web Science*, WebSci '18, New York, NY, USA, 2018. Association for Computing Machinery.
[65] Zahra Moti, Asuman Senol, Hamid Bostani, Frederik Zuiderveen Borgesius, Veelasha Moonsamy, Arunesh Mathur, and Gunes Acar. Targeted and troublesome: Tracking and advertising on children's websites. In *Procs. of the IEEE Symposium on Security and Privacy (SP)*, 2024.
[66] Doronz88. Understanding idevice protocol layers. https://github.com/doronz88/pymobiledevice3/blob/master/misc/understanding_idevice_protocol_layers.md. Accessed: March 26, 2025.
[67] Rhasspy Contributors. Piper: A fast, local neural text to speech system. https://github.com/rhasspy/piper, 2023. Accessed: March 26, 2025.
[68] Rhasspy Project. Piper TTS - en_US Amy. https://huggingface.co/rhasspy/piper-voices/tree/v1.0.0/en/en_US/amy/medium, 2023. Accessed: 2025-03-28.
[69] Bill Taylor. playsound: Pure python, cross platform, single function module with no dependencies for playing sounds. https://pypi.org/project/playsound/, 2023. Accessed: 2025-04-13.
[70] Jessica Pimienta, Jacco Brandt, Timme Bethe, Ralph Holz, Andrea Continella, Lindsay Jibb, and Quinn Grundy. Mobile apps and children's privacy: a traffic analysis of data sharing practices among children's mobile ios apps. *Archives of Disease in Childhood*, 2023.
[71] Piper Sandler Completes 46th Semi-Annual Generation Z Survey of 9,193 U.S. Teens | Piper Sandler. https://www.pipersandler.com/news/piper-sandler-completes-46th-semi-annual-generation-z-survey-9193-us-teens, April 2025. [Online; accessed 13. Apr. 2025].
[72] Johannes Buchner. Imagehash: perceptual image hashing in python. https://github.com/JohannesBuchner/imagehash, 2013. Accessed: 2025-03-28.
[73] FTC. Children's Online Privacy Protection Rule ("COPPA"). https://www.ftc.gov/node/60175, 2023. [Accessed 28 Feb. 2023].
[74] Qwen Team. Qwen2.5-vl technical report, 2025.
[75] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
[76] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
[77] Facundo Olano. App Store Scraper. https://github.com/facundoolano/app-store-scraper/tree/master. Accessed: March 24, 2025.
[78] Google. Google Ads Keyword Planner. https://ads.google.com/home/tools/keyword-planner/. Accessed: March 24, 2025.
[79] Majd Taby. ipatool: Command-line tool for downloading ios apps from the app store. https://github.com/majd/ipatool. Accessed: March 25, 2025.
[80] DuckDuckGo. Duckduckgo tracker radar. https://github.com/duckduckgo/tracker-radar, 2020.
[81] Midas Nouwens, Ilaria Liccardi, Michael Veale, David Karger, and Lalana Kagal. Dark Patterns after the GDPR: Scraping Consent Pop-ups and Demonstrating their Influence. In *Procs. of the Conference on Human Factors in Computing Systems (CHI)*, New York, NY, USA, 2020. ACM.
[82] Google Ads. Financial products and services policy, 2024. Accessed: 2025-03-30.
[83] PREDATOR IN THE WIRES: Ahmed Eltantawy Targeted with Predator Spyware After Announcing Presidential Ambitions - The Citizen Lab, January 2024. [Online; accessed 15. Apr. 2025].
[84] About Lockdown Mode - Apple Support, April 2025. [Online; accessed 12. Apr. 2025].
[85] PREDATOR IN THE WIRES: Ahmed Eltantawy Targeted with Predator Spyware After Announcing Presidential Ambitions - The Citizen Lab, January 2024. [Online; accessed 8. Sep. 2025].
[86] Fast and reliable end-to-end testing for modern web apps | Playwright Python, April 2025. [Online; accessed 12. Apr. 2025].
[87] Martin Degeling, Christine Utz, Christopher Lentzsch, Henry Hosseini, Florian Schaub, and Thorsten Holz. We value your privacy... now take some cookies: Measuring the gdpr's impact on web privacy. *arXiv preprint arXiv:1808.05096*, 2018.
[88] Pierre Laperdrix, Oleksii Starov, Quan Chen, Alexandros Kapravelos, and Nick Nikiforakis. Fingerprinting in style: Detecting browser extensions via injected style sheets. In *Procs. of the USENIX Security Symposium*, 2021.
[89] Babak Amin Azad, Oleksii Starov, Pierre Laperdrix, and Nick Nikiforakis. Web runner 2049: Evaluating third-party anti-bot services. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 17th International Conference, DIMVA 2020, Lisbon, June 24–26, 2020, Proceedings 17*, pages 135–159. Springer, 2020.
[90] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Procs. of the 2016 ACM SIGSAC conference on computer and communications security*, 2016.
[91] Inspecting iOS and iPadOS | Apple Developer Documentation, April 2025. [Online; accessed 12. Apr. 2025].
[92] Content, not 'Chrome', April 2025. [Online; accessed 12. Apr. 2025].
[93] Browse the web privately in Safari on iPhone - Apple Support, April 2025. [Online; accessed 12. Apr. 2025].
[94] Check whether a website is encrypted in Safari on iPhone, April 2025. [Online; accessed 12. Apr. 2025].
[95] Russell Graves. Analyzing iOS 16 Lockdown Mode: Browser Features and Performance. *Syonyk's Project Blog*, July 2022.
[96] Hans WA Hanley and Zakir Durumeric. Machine-made media: Monitoring the mobilization of machine-generated articles on misinformation and mainstream news websites. In *Procs. of the International AAAI Conference on Web and Social Media*, volume 18, 2024.
[97] Asuman Senol, Alisha Ukani, Dylan Cutler, and Igor Bilogrevic. The double edged sword: Identifying authentication pages and their fingerprinting behavior. In *Procs. of the ACM Web Conference 2024*, WWW '24, New York, NY, USA, 2024. Association for Computing Machinery.
[98] Eric Zeng, Tadayoshi Kohno, and Franziska Roesner. Bad news: Clickbait and deceptive ads on news and misinformation websites. In *Workshop on Technology and Consumer Protection*, pages 1–11, 2020.
[99] How To Add Icons, February 2025. [Online; accessed 12. Apr. 2025].
[100] Detect if user is running Voice Co… | Apple Developer Forums, April 2025. [Online; accessed 15. Apr. 2025].
[101] Zeyi Liao, Lingbo Mo, Chejian Xu, Mintong Kang, Jiawei Zhang, Chaowei Xiao, Yuan Tian, Bo Li, and Huan Sun. Eia: Environmental injection attack on generalist web agents for privacy leakage. *arXiv preprint arXiv:2409.11295*, 2024.
[102] FEATURE REQUEST: Implementing Tunnel Creation without sudo · Issue #1260 · doronz88/pymobiledevice3, April 2025. [Online; accessed 12. Apr. 2025].
[103] Aniketh Girish, Tianrui Hu, Vijay Prakash, Daniel J Dubois, Srdjan Matic, Danny Yuxing Huang, Serge Egelman, Joel Reardon, Juan Tapiador, David Choffnes, et al. In the room where it happens: Characterizing local communication and threats in smart homes. In *Procs. of the 2023 ACM on Internet Measurement Conference*, 2023.
[104] Shihan Deng, Weikai Xu, Hongda Sun, Wei Liu, Tao Tan, Jianfeng Liu, Ang Li, Jian Luan, Bin Wang, Rui Yan, and Shuo Shang. Mobile-bench: An evaluation benchmark for llm-based mobile agents, 2024.
[105] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. Rico: A mobile app dataset for building data-driven design applications. In *Procs. of the Annual ACM Symposium on User Interface Software and Technology*, UIST '17, New York, NY, USA, 2017. Association for Computing Machinery.
[106] OpenCV team. Opencv: Open source computer vision library. https://opencv.org/, 2024. Version 4.11.0.
[107] YourAdChoices.com | Welcome to YourAdChoices.com. https://youradchoices.com, April 2025. [Online; accessed 13. Apr. 2025].
[108] Mei Fang, GuangXue Yue, and Qingcang Yu. The study on an application of otsu method in canny operator. In *Proceedings. The 2009 International symposium on information processing (ISIP 2009)*, page 109. Citeseer, 2009.

A Appendices

A.1 Child-Related Keywords Used for App Discovery

To identify children's apps using the App Store Scraper, we compiled a list of relevant keywords. The final set of keywords used in our study includes:

dress up games for girls, *cartoons for kids*, *toddler games*, *abc games apps*, *childrens games*, *abc kids*, *painting games for kids*, *pre k games*, *childrens apps*, *educational apps*, *educational games for kids*, *preschool games*, *math games for kids*, *kids apps*, *games for kids*, *learning games for kids*, *learning games for kindergarten*, and *puzzles for kids*.

A.2 The Template Matching for Ad Detection

Advertisement Detection. We use OpenCV's [106] template matching method to detect ads that appear within mobile apps. In the app screenshots, we search for the AdChoices [107] icon and four other ad transparency icons. We compile the list of icons by manually reviewing screenshots obtained in a pilot study of 100 apps with WhisperTest. We manually review app screenshots to detect ads and extracted four more common transparency icons for use in our detection pipeline. We use a sliding window style approach to search for the icons in the app screenshots using OpenCV's [106] template matching method. At each location, the method compares the template to the corresponding region in the input image of the same size. Then, for each region, we apply the Normalized Cross-Correlation Coefficient method to find the correlation between the template and the current image region. The method computes a value between -1 and 1, where -1 is the perfect inverse, 0 is no correlation and 1 is a perfect match. We repeat this search for a scaled version of the ad icons to prevent false negatives due to size differences. We set the matching threshold to 0.5 to minimize false negatives (missed ads), since we have a human in the loop to easily remove false positives.

Before template matching, we convert both the template and image to grayscale and apply Canny edge detection using OpenCV. Following Fang et al. [108], we use Otsu's method to determine the upper threshold and set the lower threshold to half of that. These preprocessing steps simplify the data by removing color information and highlighting edges for more effective matching.

A.3 Example Usage of WhisperTest

Listing 1 shows how `WhisperTestDevice` can be used to automate key tasks such as app installation, screen recording, PCAP capture, screen parsing, and issuing voice commands on an iOS device.

A.4 Most common third-party tracker entities

Table 6 lists the most common third-party tracking entities detected across the studied children's apps. Google was by far the most prevalent, appearing in over two-thirds of the apps, followed by InMobi and Amazon; even under the reject mode.

Table 6: Most common third-party tracker entities found across studied children's apps under both consent modes. The right-hand columns indicate the number of apps in which each entity appeared.

Entity	Accept	Reject
Google	135	117
InMobi	62	50
Amazon.com	38	31
AppsFlyer	33	29
Amplitude	22	21
Smaato	9	6
PubMatic	7	5
Microsoft	5	2
Entravision	4	2
Others	43	15

A.5 Examples of Detected Ads

In Figure 5 we share two ads detected in children's apps during our automated testing.

Figure 5: Examples of ads detected in children's apps.

A.6 Navigation Examples

This section presents additional navigation examples from children's apps, showcasing how WhisperTest handles key interaction challenges. The examples include dismissing subscription dialogs, Figure 6, completing age verification flows guided by LLMs, Figure 7, and navigating cartoonish interfaces with limited accessibility, Figure 8. Each figure highlights different steps WhisperTest takes using its fallback-based interaction pipeline.

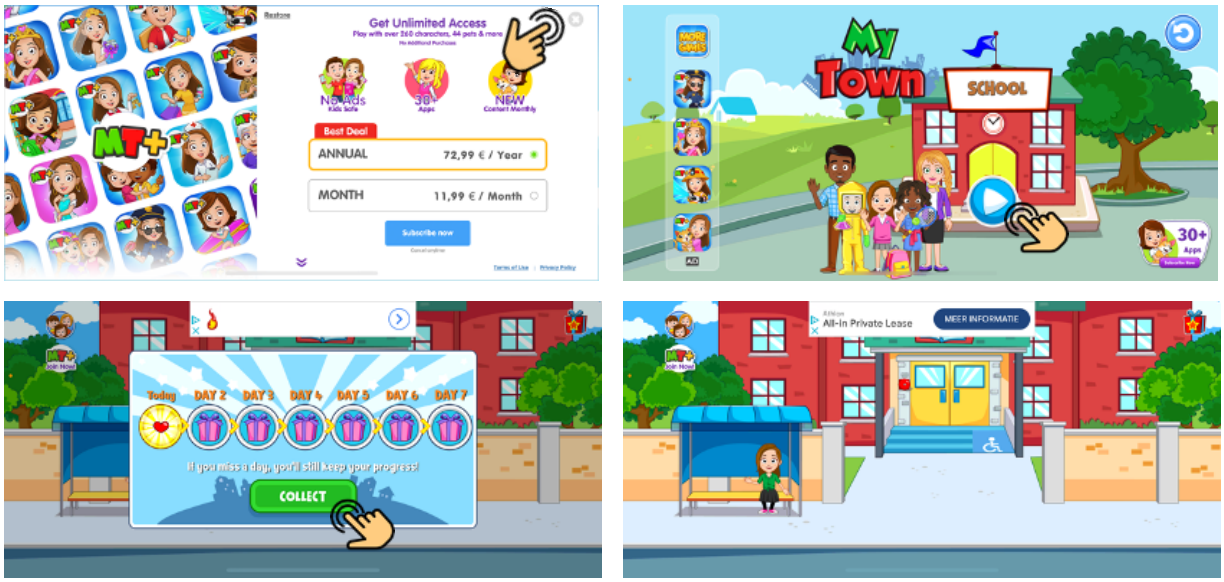


Figure 6: Example of WhisperTest's interaction with a subscription dialog shown at app launch. WhisperTest locates and taps the subtle "X" icon to dismiss the subscription dialog; then taps the cartoonish play icon, and then "Collect" to access the main game interface where an ad is visible.

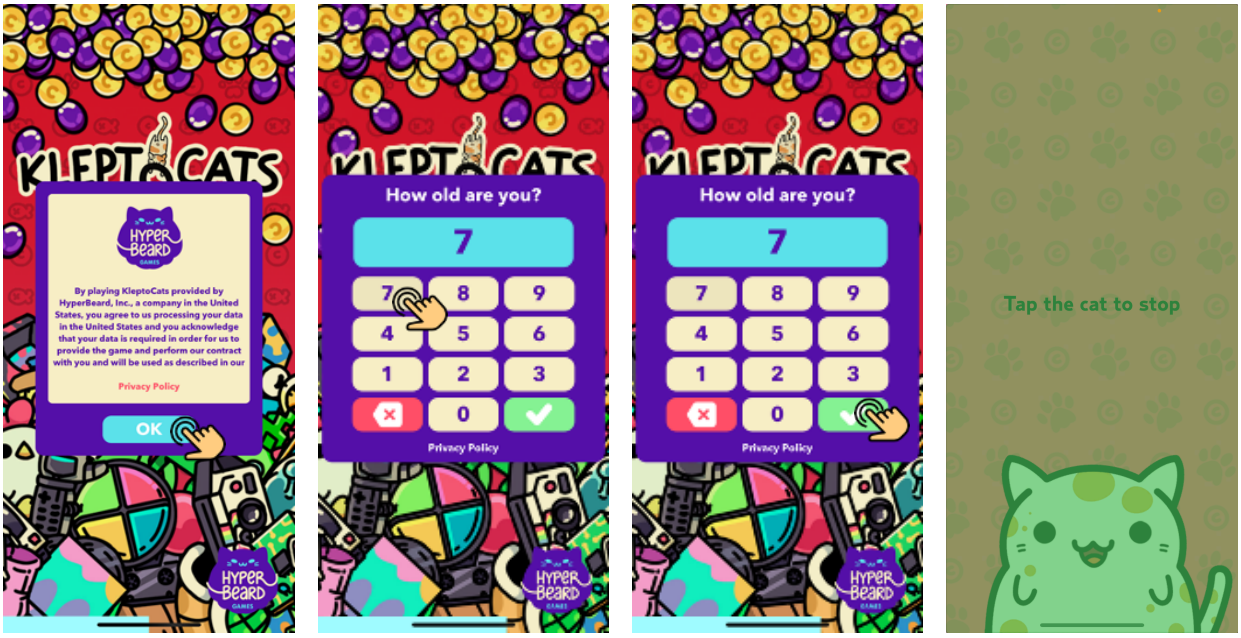


Figure 7: Example of WhisperTest automatically interacting with an age verification dialog, based on LLM guidance. It first accepts the privacy policy, then enters "7" as the age following LLM's response, since the LLM is prompted to imagine being a 7-year-old (Table 8). It finally taps the check mark to access the app content.

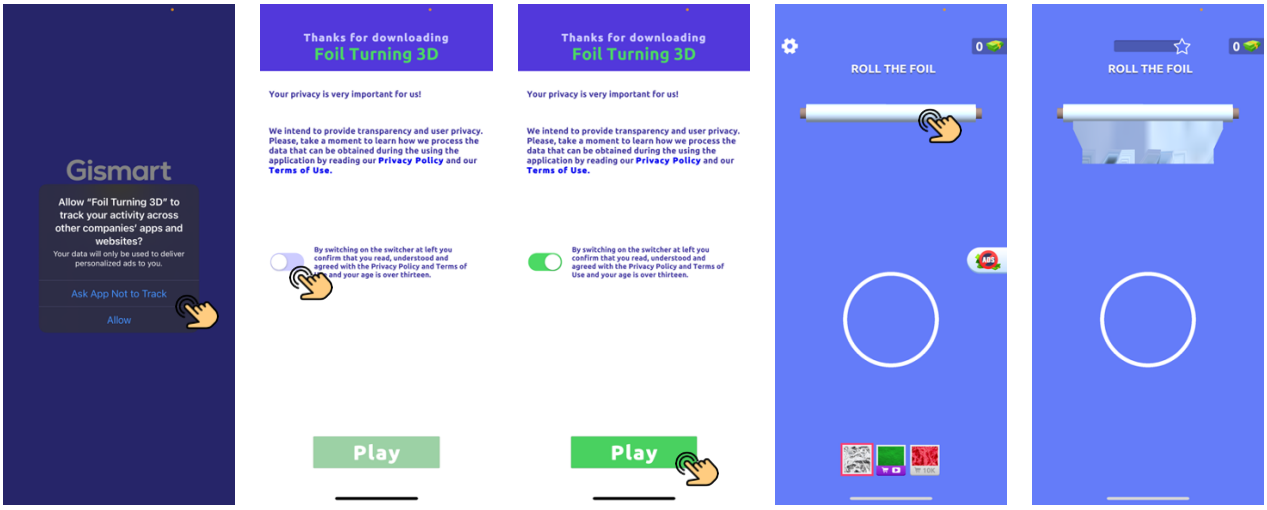


Figure 8: Example of WhisperTest navigating a children's app: it taps "Ask App Not to Track", toggles the terms of service button, taps Play, and interacts with the correct element to start the game.

A.7 LLM Prompts for Navigation and Consent Detection

This section outlines the prompts used by WhisperTest to guide navigation and consent detection during app interaction. First, a dedicated consent detection prompt, displayed in Table 7 is issued to determine whether the current screen contains a privacy or cookie consent dialog. If such a dialog is detected, corresponding consent-handling instructions (e.g., accept or reject) are dynamically appended to the navigation prompt. The navigation prompt, shown in Table 8 includes detailed guidelines for tasks like handling subscription offers, completing verification steps, etc. It also specifies the expected JSON output format and a step-by-step reasoning procedure.

Model Selection Pilot Study. To detect consent dialogs (§ 4.4), we used `Qwen2.5-7B`. Although many open-source models can be used for this task, a small evaluation on 100 samples showed `Qwen2.5-7B` achieving 98% accuracy. Once a consent screen is detected, `Qwen2.5-14B` is used to generate an appropriate response based on the specified consent mode (e.g., Accept or Reject). In a test on 50 randomly selected apps, this approach successfully interacted with 48 consent dialogs. During the pilot study, we also compared models for handling age verification screens, subscription prompts, and consent dialogs. `Qwen2.5-14B` proved optimal, demonstrating strong instruction-following ability and faster execution times than the alternatives.

Table 7: Prompt used to detect the presence of cookie consent dialogs.

Prompt for Consent Detection:
You are an AI assistant specialized in detecting privacy/cookie consent dialogs in mobile applications.
Your task is to analyze the provided screen elements and determine if the screen contains a consent dialog.
Look for key indicators such as links to privacy policies, accept/reject buttons, or keywords such as GDPR, personal data, privacy, advertising, etc.
If you detect that the screen is a consent dialog, output the JSON: `{"consent_dialog": "yes"}`.
If there's no mention of cookies or if it only includes terms of service, output: `{"consent_dialog": "no"}`.
Ensure your response is valid JSON and contains no additional text.
`Screen elements: [Placeholder for actual screen elements]`

Table 8: The prompt used for the LLM to generate navigation actions based on textual screen data. The image-based model uses a different output format, which includes a bounding box field: `"bbox_2d": [x1, y1, x2, y2]`.

System Prompt: You are an AI assistant specialized in navigating mobile applications designed for children. Your primary goal is to simulate natural, human-like interactions by processing a text-based representation of the current screen and determining the optimal next action to guide the user through the app.

User Prompt:

Objective: Analyze the text content of the current mobile screen and determine the single best next action that a real user would intuitively take to navigate the app.

Guidelines:

(1) **Single Best Action:** Choose the one action that a real user would intuitively take next.

(2) **Use Provided Data Only:** Rely only on the visible screen elements and the user's goal. Do not infer extra information.

(3) **Track Action History:** If the same action is repeated more than twice based on the "Action History", try a different element.

(4) **Action Formats:** `Tap [screen element]` or `Type [value]`

(5) **Typing Procedure:** When inputting text to fill in a form, first tap the target field to focus it, then in the next step, type the desired value.

(6) **Identify Clickable Elements:** Consider an element clickable if its text or description indicates interactivity. Look for actionable cues (e.g., 'dismiss', 'close', 'start', 'play') or phrases like 'a completed task or confirmation' or 'a play button for a video game' that imply a clickable control. Make sure to scan the entire list of screen elements—even those at the very end—to capture all potential actionable items.

(7) **Data Source Note:** The screen data may include both visible text and descriptions of icons (which may not have any text). Treat icon descriptions as valid if they include clear actionable cues.

(8) **Subscription/Premium:** If prompted to subscribe or upgrade, your goal here is to get out of this subscription page as quickly as possible. Prioritize elements that dismiss or bypass the offer window dialog and AVOID options that lead to subscriptions, free trials, or payments such as 'start a free trial', 'cancel any time', 'restore purchase', continue, try, etc.

(9) **Verification Challenges:** If a verification step (e.g., age or math) appears, first try to dismiss or close the pop-up or dialog. Exiting unwanted screens quickly is preferable. Only complete the verification if it cannot be bypassed by tapping on the desired screen element. To do so, for age verification, assume the user is 7.

If a consent dialog is detected, append the relevant consent-handling instructions.

• **Consent & Agreement Handling (Reject):** When a data processing consent dialog or screen appears, your goal always is to tap on Reject or Decline button. You must avoid closing or exiting the consent dialog without rejection. If there is no explicit 'Reject' button, first tap 'Manage options' or 'More info' or similar options to adjust your choices. Then choose reject button or confirm default choices by tapping on Confirm choices button or similar buttons. Remember, you are not allowed to proceed through the app without rejecting the consent.

• **Consent & Agreement Handling (Accept):** When a data processing consent dialog or screen appears, your goal always is to tap on Accept button. You must avoid closing or exiting the consent dialog without acceptance.

Examples:

`[Placeholder for actual examples]`

History of Actions:

Pay attention to the history of actions and the success of each action to track the progress and avoid stucking in a loop. Avoid repeating actions that have already been tried multiple times and failed.

`Action history: [Placeholder for actual history of actions]`

Step-by-Step Reasoning:

(1) **Determine Page Type:** Identify the page's purpose from the screen text (e.g., login, subscription, age verification, consent, etc).

(2) **Extract Clickable Elements:** Identify and list all clickable elements based on guideline 7 and 8, while reviewing history of actions.

(3) **Filter Extraneous Items:** Remove any non-actionable or redundant elements to ensure all valid options are considered—even those at the end of a long list.

(4) **Compare Options:** Evaluate the elements against the guidelines, focusing on bypassing unwanted processes and moving toward the main page.

(5) **Choose Best Action:** Select the single, most intuitive action a user would take next (tap or type). If no appropriate action is available, respond with `NO OPTION AVAILABLE`.

(6) **Output JSON:** Output the next action name and bounding box coordinates (x1, y1, x2, y2) in JSON format following the *EXAMPLE OUTPUT FORMAT*

Always think step by step, and clearly delineate your reasoning for Steps 1, 2, 3, 4, 5, and 6.

Output Format:

```
{"action":"", "value": "", ("bbox_2d": [x1, y1, x2, y2])}
{"action":"", "screen_element":"", ("bbox_2d": [x1, y1, x2, y2])}
```

Your response should strictly adhere to the guidelines and process steps above. Navigate naturally and efficiently toward the app's main page, and continue until an advertisement appears.

```
Screen elements: [Placeholder for actual screen elements]
```

A.8 Web Measurements

Table 9 shows the most prevalent tracking domains we found in the web measurement study on reliable and unreliable websites in Lockdown mode.

Figure 10 shows the differences in web fonts between non-lockdown mode and lockdown mode. We see certain web fonts rendered as empty rectangles due to lockdown mode.

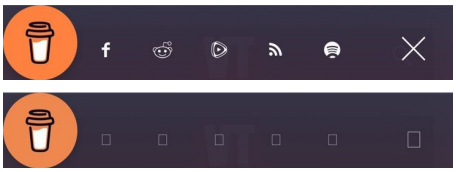


Figure 10: Menu items that use web fonts rendered as empty rectangles due to iPhone's Lockdown mode [84].

Figure 9 shows the Page Menu on the left, and the two scraped pages on the right. The Privacy Report (middle) displays the tracking domains and entities detected by Safari. The Connection Security Details page displays the certificate details.

Table 9: Most prevalent tracking domains found in the web measurement study on reliable and unreliable websites in Lockdown mode. Based on 40 websites in each category.

Entity (Reliable sites)	Num. sites	Entity (Unreliable sites)	Num. sites
doubleclick.net	28	google.com	24
googletagmanager.com	26	google-analytics.com	22
google-analytics.com	24	googletagmanager.com	18
google.com	24	googlesyndication.com	11
googlesyndication.com	12	doubleclick.net	10
cloudflare.com	12	youtube.com	6
criteo.com	10	adsco.re	5
amazon-adsystem.com	10	onesignal.com	5
pubmatic.com	9	facebook.net	4
id5-sync.com	8	jsdelivr.net	4

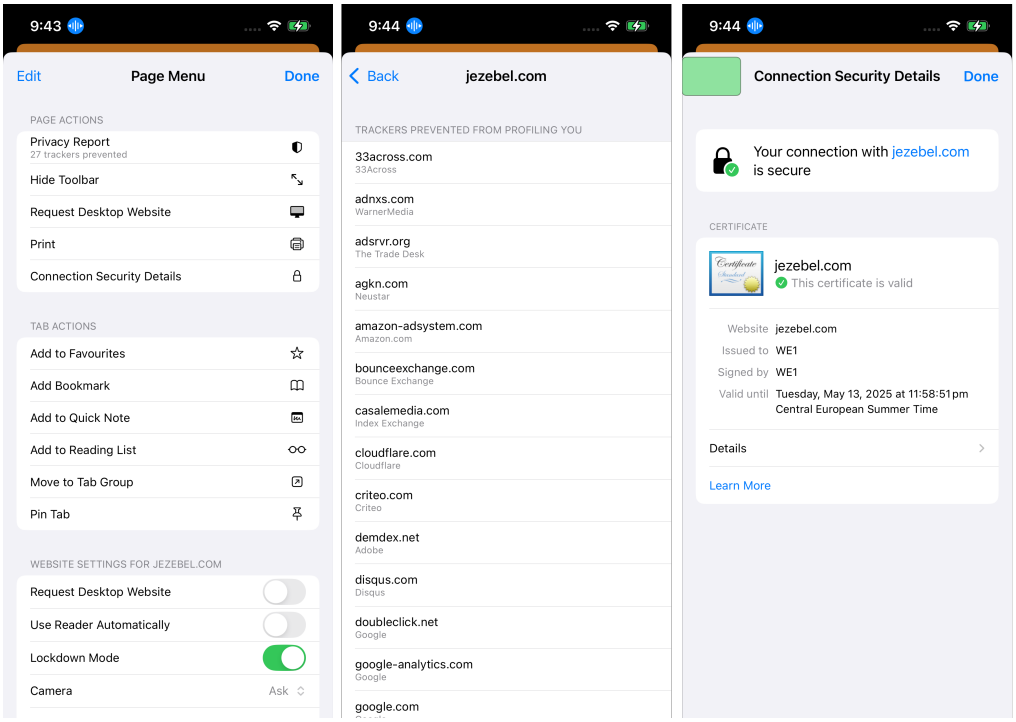


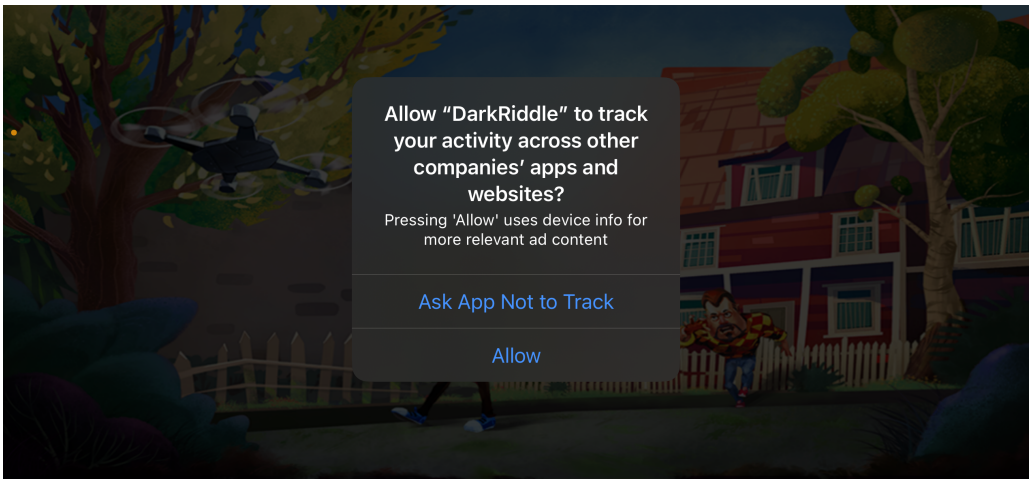
Figure 9: Safari pages scraped in the web measurement. The green rectangle on the top left of the rightmost screenshot is an artifact of the accessibility audit that we use to scrape the page contents.

```python
from whisper_test.device import WhisperTestDevice

app_bundle_id = "com.example.test"
ipa_path = "path/to/ipa/file.ipa"
pcap_path = "path/to/capture.pcap"

device = WhisperTestDevice()

device.install_app_via_ipa(ipa_path)
device.launch_app(app_bundle_id)

device.start_screen_recording()
device.start_pcap_capture(pcap_path)

scr_shot, __ = device.take_screenshots(app_bundle_id)

print("Getting screen content by accessibility...")
a11y = device.get_screen_content_by_a11y()
print("Getting screen content by OCR...")
ocr = device.get_screen_content_by_ocr(sshot)
# the caller code can decide what to do here, given the screen contents

print("Issuing voice command to go home...")
device.say("Go home")

device.uninstall_app(app_bundle_id)
device.stop_pcap_capture()
device.stop_screen_recording()
```

Listing 1: Example usage of `WhisperTestDevice`.

A.9 Example of Screen Interpretation via Accessibility and OCR

Figure 11: Screenshot of an App Tracking Transparency (ATT) prompt shown in the app, used to demonstrate how WhisperTest interprets screen content via accessibility and OCR.



Accessibility Data:

```
[
 ['Allow 'DarkRiddle' to track your activity across
  other companies' apps and websites?'],
 ['Pressing "Allow" uses device info for more relevant
  ad content'],
 ['Ask App Not to Track', Button],
 ['Allow', Button]
]
```

OmniParser OCR Output:

```
[
 ['0', 'text', 'Allow "DarkRiddle" to track',
  '941', '239', '660', '60'],
 ['1', 'icon', 'Ask Not to Track App ',
  '856', '664', '821', '139'],
 ['2', 'icon', 'Allow ',
  '856', '798', '821', '136'],
 ['3', 'text', "your activity across other companies' apps
  and websites? Pressing 'Allow' uses device info for
  more relevant ad content ",
  '857', '278', '815', '392']
]
```